VISIT…

LANZAROTE
Caliente.com

Professional
ASP.NET 3.5
In C# and VB

Bill Evjen
Scott Hanselman
Devin Rader

WILEY

Wiley Publishing, Inc.

Professional ASP.NET 3.5
In C# and VB

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
`www.wiley.com`

Copyright © 2008 by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-0-470-18757-9

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data is available from the publisher.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Legal Department, Wiley Publishing, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, (317) 572-3447, fax (317) 572-4355, or online at `http://www.wiley.com/go/permissions`.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Website is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or Website may provide or recommendations it may make. Further, readers should be aware that Internet Websites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Trademarks: Wiley, the Wiley logo, Wrox, the Wrox logo, Wrox Programmer to Programmer, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

About the Authors

Bill Evjen is an active proponent of .NET technologies and community-based learning initiatives for .NET. He has been actively involved with .NET since the first bits were released in 2000. In the same year, Bill founded the St. Louis .NET User Group (`www.stlnet.org`), one of the world's first such groups. Bill is also the founder and former executive director of the International .NET Association (`www.ineta.org`), which represents more than 500,000 members worldwide.

Based in St. Louis, Missouri, USA, Bill is an acclaimed author and speaker on ASP.NET and XML Web Services. He has authored or co-authored more than fifteen books including *Professional C# 2008, Professional VB 2008, ASP.NET Professional Secrets, XML Web Services for ASP.NET*, and *Web Services Enhancements: Understanding the WSE for Enterprise Applications* (all published by Wiley Publishing, Inc.). In addition to writing, Bill is a speaker at numerous conferences, including DevConnections, VSLive, and TechEd. Along with these items, Bill works closely with Microsoft as a Microsoft Regional Director and an MVP.

Bill is the Technical Architect for Lipper (`www.lipperweb.com`), a wholly-owned subsidiary of Reuters, the international news and financial services company. He graduated from Western Washington University in Bellingham, Washington, with a Russian language degree. When he isn't tinkering on the computer, he can usually be found at his summer house in Toivakka, Finland. You can reach Bill at `evjen@yahoo.com`.

Scott Hanselman works for Microsoft as a Senior Program Manager in the Developer Division, aiming to spread the good word about developing software, most often on the Microsoft stack. Before this he worked in eFinance for 6+ years and before that he was a Principal Consultant at a Microsoft Partner for nearly 7 years. He was also involved in a few things like the MVP and RD programs and will speak about computers (and other passions) whenever someone will listen to him. He blogs at `http://www.hanselman.com` and podcasts at `http://www.hanselminutes.com` and contributes to `http://www.asp.net`, `http://www.windowsclient.net`, and `http://www.silverlight.net`.

Devin Rader is a Product Manager on the Infragistics Web Client team, responsible for leading the creation of Infragistics ASP.NET and Silverlight products. Devin is also an active proponent and member of the .NET developer community, being a co-founder of the St. Louis .NET User Group, an active member of the New Jersey .NET User Group, a former board member of the International .NET Association (INETA), and a regular speaker at user groups. He is also a contributing author on the Wrox title *Silverlight 1.0* and a technical editor for several other Wrox publications and has written columns for ASP.NET Pro magazine, as well as .NET technology articles for MSDN Online. You can find more of Devin's musings at `www.geekswithblogs.com/devin`.

Credits

Acquisitions Director
Jim Minatel

Development Editors
Adaobi Obi Tulton
Sydney Jones

Technical Editors
Eric Engler
Alexei Gorkov
Doug Holland
Darren Kindberg
Mark Strawmeyr

Production Editor
Angela Smith

Copy Editors
Nancy Rapoport
Sydney Jones

Editorial Manager
Mary Beth Wakefield

Production Manager
Tim Tate

Vice President and Executive Group Publisher
Richard Swadley

Vice President and Executive Publisher
Joseph B. Wikert

Project Coordinator, Cover
Lynsey Stanford

Proofreader
Sossity Smith

Indexer
J & J Indexing

Acknowledgments

I have said it before, and I will say it again: Writing a book may seem like the greatest of solo endeavors, but it requires a large team of people working together to get technical books out the door-and this book is no exception. First and foremost, I would like to thank Jim Minatel of Wrox for giving me the opportunity to write the original ASP.NET book, which then led to this special edition. There is nothing better than getting the opportunity to write about your favorite topic for the world's best publisher!

Besides Jim, I worked with the book's development editor, Adaobi Obi Tulton. Adaobi kept the book moving along even with all the interruptions coming our way. Without Adaobi's efforts, this book would not have happened.

I worked closely with both Scott Hanselman and Devin Rader on this book, and these guys deserve a lot of thanks. I appreciate your help and advice throughout the process. Thanks guys!

I would also like to thank the various editors who worked on this book: Alexei Gorkov, Mark Strawmeyr, Darren Kindberg, Eric Engler, and Doug Holland. Big and ongoing thanks go to the Wrox/Wiley gang including Joe Wikert (publisher), Katie Mohr (acquisitions editor), and David Mayhew (marketing).

Finally, thanks to my entire family. Book writing is a devil in disguise as it is something that I love to do but at the same time, takes way too much time away from my family. Thanks to my family for putting up with this and for helping me get these books out the door. I love you all.

— Bill Evjen

Contents

Introduction **xxxi**

Chapter 1: Application and Page Frameworks 1

Application Location Options **1**
Built-In Web Server 2
IIS 3
FTP 4
Web Site Requiring FrontPage Extensions 5
The ASP.NET Page Structure Options **6**
Inline Coding 8
Code-Behind Model 10
ASP.NET 3.5 Page Directives **13**
@Page 14
@Master 17
@Control 18
@Import 19
@Implements 21
@Register 21
@Assembly 22
@PreviousPageType 22
@MasterType 23
@OutputCache 23
@Reference 24
ASP.NET Page Events **24**
Dealing with PostBacks **26**
Cross-Page Posting **27**
ASP.NET Application Folders **33**
\App_Code Folder 33
\App_Data Folder 38
\App_Themes Folder 38
\App_GlobalResources Folder 39
\App_LocalResources 39
\App_WebReferences 39
\App_Browsers 39

Compilation **40**
Build Providers **44**
Using the Built-in Build Providers 45
Using Your Own Build Providers 46
Global.asax **51**
Working with Classes Through VS2008 **54**
Summary **61**

Chapter 2: ASP.NET Server Controls and Client-Side Scripts 63

ASP.NET Server Controls **63**
Types of Server Controls 64
Building with Server Controls 65
Working with Server Control Events 67
Applying Styles to Server Controls **70**
Examining the Controls' Common Properties 70
Changing Styles Using Cascading Style Sheets 72
HTML Server Controls **76**
Looking at the HtmlControl Base Class 79
Looking at the HtmlContainerControl Class 80
Looking at All the HTML Classes 80
Using the HtmlGenericControl Class 81
Manipulating Pages and Server Controls with JavaScript **83**
Using Page.ClientScript.RegisterClientScriptBlock 84
Using Page.ClientScript.RegisterStartupScript 86
Using Page.ClientScript.RegisterClientScriptInclude 88
Client-Side Callback **89**
Comparing a Typical Postback to a Callback 89
Using the Callback Feature — A Simple Approach 90
Using the Callback Feature with a Single Parameter 96
Using the Callback Feature — A More Complex Example 99
Summary **105**

Chapter 3: ASP.NET Web Server Controls 107

An Overview of Web Server Controls **107**
The Label Server Control **108**
The Literal Server Control **110**
The TextBox Server Control **111**
Using the Focus() Method 112
Using AutoPostBack 113
Using AutoCompleteType 114

The Button Server Control **115**
The CausesValidation Property 115
The CommandName Property 115
Buttons That Work with Client-Side JavaScript 117
The LinkButton Server Control **119**
The ImageButton Server Control **119**
The HyperLink Server Control **120**
The DropDownList Server Control **121**
Visually Removing Items from a Collection **124**
The ListBox Server Control **125**
Allowing Users to Select Multiple Items 126
An Example of Using the ListBox Control 126
Adding Items to a Collection 129
The CheckBox Server Control **129**
How to Determine Whether Check Boxes Are Checked 131
Assigning a Value to a Check Box 131
Aligning Text Around the Check Box 131
The CheckBoxList Server Control **132**
The RadioButton Server Control **134**
The RadioButtonList Server Control **136**
Image Server Control **138**
Table Server Control **139**
The Calendar Server Control **142**
Making a Date Selection from the Calendar Control 142
Choosing a Date Format to Output from the Calendar 144
Making Day, Week, or Month Selections 144
Working with Date Ranges 144
Modifying the Style and Behavior of Your Calendar 147
AdRotator Server Control **151**
The Xml Server Control **153**
Panel Server Control **153**
The PlaceHolder Server Control **156**
BulletedList Server Control **157**
HiddenField Server Control **162**
FileUpload Server Control **164**
Uploading Files Using the FileUpload Control 164
Giving ASP.NET Proper Permissions to Upload Files 167
Understanding File Size Limitations 167
Uploading Multiple Files from the Same Page 170
Placing the Uploaded File into a Stream Object 172
Moving File Contents from a Stream Object to a Byte Array 173

MultiView and View Server Controls **174**
Wizard Server Control **178**
  Customizing the Side Navigation 180
  Examining the AllowReturn Attribute 180
  Working with the StepType Attribute 180
  Adding a Header to the Wizard Control 181
  Working with the Wizard's Navigation System 182
  Utilizing Wizard Control Events 183
  Using the Wizard Control to Show Form Elements 184
ImageMap Server Control **189**
Summary **192**

Chapter 4: Validation Server Controls **193**

Understanding Validation **193**
Client-Side versus Server-Side Validation **194**
ASP.NET Validation Server Controls **195**
  Validation Causes 196
  The RequiredFieldValidator Server Control 197
  The CompareValidator Server Control 202
  The RangeValidator Server Control 206
  The RegularExpressionValidator Server Control 209
  The CustomValidator Server Control 211
  The ValidationSummary Server Control 216
Turning Off Client-Side Validation **220**
Using Images and Sounds for Error Notifications **221**
Working with Validation Groups **222**
Summary **227**

Chapter 5: Working with Master Pages **229**

Why Do You Need Master Pages? **229**
The Basics of Master Pages **231**
Coding a Master Page **233**
Coding a Content Page **235**
  Mixing Page Types and Languages 239
  Specifying Which Master Page to Use 241
  Working with the Page Title 242
  Working with Controls and Properties from the Master Page 243
Specifying Default Content in the Master Page **250**
Programmatically Assigning the Master Page **251**
Nesting Master Pages **253**
Container-Specific Master Pages **257**

Event Ordering **258**
Caching with Master Pages **259**
ASP.NET AJAX and Master Pages **259**
Summary **262**

Chapter 6: Themes and Skins 263

Using ASP.NET Themes **263**
- Applying a Theme to a Single ASP.NET Page 263
- Applying a Theme to an Entire Application 265
- Removing Themes from Server Controls 266
- Removing Themes from Web Pages 267
- Understanding Themes When Using Master Pages 267
- Understanding the StyleSheetTheme Attribute 268

Creating Your Own Themes **268**
- Creating the Proper Folder Structure 268
- Creating a Skin 269
- Including CSS Files in Your Themes 272
- Having Your Themes Include Images 275

Defining Multiple Skin Options **278**
Programmatically Working with Themes **280**
- Assigning the Page's Theme Programmatically 280
- Assigning a Control's SkinID Programmatically 281

Themes, Skins, and Custom Controls **281**
Summary **286**

Chapter 7: Data Binding in ASP.NET 3.5 287

Data Source Controls **287**
- SqlDataSource Control 289
- LINQ Data Source Control 302
- AccessDataSource Control 307
- XmlDataSource Control 307
- ObjectDataSource Control 309
- SiteMapDataSource Control 314

Configuring Data Source Control Caching **314**
Storing Connection Information **315**
Using Bound List Controls with Data Source Controls **317**
- GridView 318
- Editing GridView Row Data 334
- Deleting GridView Data 341
- DetailsView 344
- Inserting, Updating, and Deleting Data Using DetailsView 349

ListView 350
FormView 360
Other Databound Controls 365
DropDownList, ListBox, RadioButtonList, and CheckBoxList 365
TreeView 366
Ad Rotator 366
Menu 367
Inline Data-Binding Syntax 367
Data-Binding Syntax Changes 368
XML Data Binding 369
Expressions and Expression Builders 369
Summary 375

Chapter 8: Data Management with ADO.NET 377

Basic ADO.NET Features 378
Common ADO.NET Tasks 378
Basic ADO.NET Namespaces and Classes 383
Using the Connection Object 384
Using the Command Object 386
Using the DataReader Object 387
Using Data Adapter 389
Using Parameters 392
Understanding DataSet and DataTable 395
Using Oracle as Your Database with ASP.NET 3.5 400
The DataList Server Control 403
Looking at the Available Templates 403
Working with ItemTemplate 404
Working with Other Layout Templates 407
Working with Multiple Columns 409
The ListView Server Control 410
Looking at the Available Templates 410
Using the Templates 411
Creating the Layout Template 412
Creating the ItemTemplate 414
Creating the EditItemTemplate 415
Creating the EmptyItemTemplate 415
Creating the InsertItemTemplate 416
The Results 416
Using Visual Studio for ADO.NET Tasks 419
Creating a Connection to the Data Source 419
Working with a Dataset Designer 422
Using the CustomerOrders DataSet 427

Asynchronous Command Execution **432**
Asynchronous Connections 454
Summary **454**

Chapter 9: Querying with LINQ 455

LINQ to Objects **455**
Traditional Query Methods 455
Replacing Traditional Queries with LINQ 464
Data Grouping 472
Other LINQ Operators 473
LINQ Joins 473
Paging Using LINQ 475
LINQ to XML **476**
Joining XML Data 479
LINQ to SQL **481**
Insert, Update, and Delete Queries through LINQ 490
Extending LINQ 494
Summary **495**

Chapter 10: Working with XML and LINQ to XML 497

The Basics of XML **498**
The XML InfoSet 500
XSD–XML Schema Definition 501
Editing XML and XML Schema in Visual Studio 2008 503
XmlReader and XmlWriter **506**
Using XDocument Rather Than XmlReader 508
Using Schema with XmlTextReader 509
Validating Against a Schema Using an XDocument 511
Including NameTable Optimization 513
Retrieving .NET CLR Types from XML 515
ReadSubtree and XmlSerialization 517
Creating CLR Objects from XML with LINQ to XML 518
Creating XML with XmlWriter 519
Creating XML with LINQ for XML 522
Improvements for XmlReader and XmlWriter in 2.0 524
XmlDocument and XPathDocument **525**
Problems with the DOM 525
XPath, the XPathDocument, and XmlDocument 525
DataSets **530**
Persisting DataSets to XML 530
XmlDataDocument 531

The XmlDataSource Control **533**
XSLT **537**
XslCompiledTransform 539
XSLT Debugging 543
Databases and XML **544**
FOR XML AUTO 545
SQL Server 2005 and the XML Data Type 549
Summary **556**

Chapter 11: IIS7 **557**

Modular Architecture of IIS7 **557**
IIS-WebServer 558
IIS-WebServerManagementTools 561
IIS-FTPPublishingService 562
Extensible Architecture of IIS7 **562**
IIS7 and ASP.NET Integrated Pipeline **562**
Building a Customized Web Server **564**
Update Dependencies 565
Installing IIS7 on Windows Vista 565
Installing IIS7 on Windows Server 2008 565
Command-Line Setup Options 567
Unattended Setup Option 568
Upgrade 569
Internet Information Services (IIS) Manager **569**
Application Pools 570
Web Sites 575
Hierarchical Configuration 577
Delegation 581
Moving an Application from IIS6 to IIS7 **584**
Summary **586**

Chapter 12: Introduction to the Provider Model **587**

Understanding the Provider **588**
The Provider Model in ASP.NET 3.5 **589**
Setting Up Your Provider to Work with Microsoft SQL Server 7.0, 2000, 2005, or 2008 591
Membership Providers 598
Role Providers 602
The Personalization Provider 606
The SiteMap Provider 608

SessionState Providers 609
Web Event Providers 612
Configuration Providers 620
The WebParts Provider 623
Configuring Providers 625
Summary 626

Chapter 13: Extending the Provider Model 627

Providers Are One Tier in a Larger Architecture 627
Modifying Through Attribute-Based Programming 628
Simpler Password Structures Through the SqlMembershipProvider 629
Stronger Password Structures Through the SqlMembershipProvider 632
Examining ProviderBase 633
Building Your Own Providers 635
Creating the CustomProviders Application 635
Constructing the Class Skeleton Required 636
Creating the XML User Data Store 640
Defining the Provider Instance in the web.config File 641
Not Implementing Methods and Properties of the MembershipProvider Class 642
Implementing Methods and Properties of the MembershipProvider Class 643
Using the XmlMembershipProvider for User Login 651
Extending Pre-Existing Providers 652
Limiting Role Capabilities with a New LimitedSqlRoleProvider Provider 652
Using the New LimitedSqlRoleProvider Provider 656
Summary 660

Chapter 14: Site Navigation 661

XML-Based Site Maps 662
SiteMapPath Server Control 664
The PathSeparator Property 666
The PathDirection Property 668
The ParentLevelsDisplayed Property 669
The ShowToolTips Property 669
The SiteMapPath Control's Child Elements 670
TreeView Server Control 670
Identifying the TreeView Control's Built-In Styles 674
Examining the Parts of the TreeView Control 676
Binding the TreeView Control to an XML File 676
Selecting Multiple Options in a TreeView 679

- Specifying Custom Icons in the TreeView Control 683
- Specifying Lines Used to Connect Nodes 685
- Working with the TreeView Control Programmatically 687

Menu Server Control 693

- Applying Different Styles to the Menu Control 694
- Menu Events 700
- Binding the Menu Control to an XML File 701

SiteMap Data Provider 703

- ShowStartingNode 703
- StartFromCurrentNode 704
- StartingNodeOffset 705
- StartingNodeUrl 706

SiteMap API 706

URL Mapping 709

Sitemap Localization 710

- Structuring the Web.sitemap File for Localization 710
- Making Modifications to the Web.config File 711
- Creating Assembly Resource (.resx) Files 712
- Testing the Results 712

Security Trimming 714

- Setting Up Role Management for Administrators 715
- Setting Up the Administrators' Section 716
- Enabling Security Trimming 718

Nesting SiteMap Files 720

Summary 722

Chapter 15: Personalization 723

The Personalization Model 723

Creating Personalization Properties 725

- Adding a Simple Personalization Property 725
- Using Personalization Properties 726
- Adding a Group of Personalization Properties 730
- Using Grouped Personalization Properties 731
- Defining Types for Personalization Properties 731
- Using Custom Types 732
- Providing Default Values 735
- Making Personalization Properties Read-Only 735

Anonymous Personalization 735

- Enabling Anonymous Identification of the End User 736
- Working with Anonymous Identification 739
- Anonymous Options for Personalization Properties 739
- Warnings about Anonymous User Profile Storage 740

Programmatic Access to Personalization 741
Migrating Anonymous Users 741
Personalizing Profiles 743
Determining Whether to Continue with Automatic Saves 744
Personalization Providers 745
Working with SQL Server Express Edition 745
Working with Microsoft's SQL Server 7.0/2000/2005/2008 746
Using Multiple Providers 748
Managing Application Profiles 749
Properties of the ProfileManger Class 750
Methods of the ProfileManager Class 750
Building the ProfileManager.aspx Page 751
Examining the Code of ProfileManager.aspx Page 754
Running the ProfileManager.aspx Page 755
Summary 755

Chapter 16: Membership and Role Management 757

Authentication 758
Authorization 758
ASP.NET 3.5 Authentication 758
Setting Up Your Web Site for Membership 758
Adding Users 761
Asking for Credentials 776
Working with Authenticated Users 784
Showing the Number of Users Online 786
Dealing with Passwords 788
ASP.NET 3.5 Authorization 793
Using the LoginView Server Control 793
Setting Up Your Web Site for Role Management 796
Adding and Retrieving Application Roles 799
Deleting Roles 801
Adding Users to Roles 802
Getting All the Users of a Particular Role 803
Getting All the Roles of a Particular User 805
Removing Users from Roles 805
Checking Users in Roles 806
Understanding How Roles Are Cached 807
Using the Web Site Administration Tool 809
Public Methods of the Membership API 809
Public Methods of the Roles API 810
Summary 810

Chapter 17: Portal Frameworks and Web Parts 811

Introducing Web Parts 811
Building Dynamic and Modular Web Sites 813
Introducing the WebPartManager Control 813
Working with Zone Layouts 814
Understanding the WebPartZone Control 817
Allowing the User to Change the Mode of the Page 820
Modifying Zones 833
Working with Classes in the Portal Framework 841
Creating Custom Web Parts 844
Connecting Web Parts 850
Building the Provider Web Part 851
Building the Consumer Web Part 854
Connecting Web Parts on an ASP.NET Page 856
Understanding the Difficulties in Dealing with Master Pages When Connecting Web Parts 858
Summary 860

Chapter 18: HTML and CSS Design with ASP.NET 861

Caveats 862
HTML and CSS Overview 862
Introducing CSS 863
Creating Style Sheets 863
CSS Rules 866
CSS Inheritance 875
Element Layout and Positioning 876
Working with HTML and CSS in Visual Studio 884
ASP.NET 2.0 CSS–Friendly Control Adapters 893
Summary 893

Chapter 19: ASP.NET AJAX 895

Understanding the Need for AJAX 895
Before AJAX 896
AJAX Changes the Story 897
ASP.NET AJAX and Visual Studio 2008 899
Client-Side Technologies 900
Server-Side Technologies 900
Developing with ASP.NET AJAX 901
ASP.NET AJAX Applications 902
Building a Simple ASP.NET Page Without AJAX 904
Building a Simple ASP.NET Page with AJAX 906

ASP.NET AJAX's Server-Side Controls **911**
The ScriptManager Control 912
The ScriptManagerProxy Control 914
The Timer Control 916
The UpdatePanel Control 917
The UpdateProgress Control 922
Using Multiple UpdatePanel Controls **925**
Summary **928**

Chapter 20: ASP.NET AJAX Control Toolkit 929

Downloading and Installing **929**
New Visual Studio Templates 931
Adding the New Controls to the VS2008 Toolbox 932
The ASP.NET AJAX Controls **934**
ASP.NET AJAX Control Toolkit Extenders 937
AlwaysVisibleControlExtender 937
AnimationExtender 939
AutoCompleteExtender 941
CalendarExtender 944
CollapsiblePanelExtender 946
ConfirmButtonExtender and ModalPopupExtender 947
DragPanelExtender 950
DropDownExtender 951
DropShadowExtender 953
DynamicPopulateExtender 956
FilteredTextBoxExtender 959
HoverMenuExtender 961
ListSearchExtender 962
MaskedEditExtender and MaskedEditValidator 964
MutuallyExclusiveCheckBoxExtender 967
NumericUpDownExtender 968
PagingBulletedListExtender 969
PopupControlExtender 970
ResizableControlExtender 972
RoundedCornersExtender 975
SliderExtender 976
SlideShowExtender 977
TextBoxWatermarkExtender 979
ToggleButtonExtender 982
UpdatePanelAnimationExtender 983
ValidatorCalloutExtender 984

ASP.NET AJAX Control Toolkit Server Controls **985**
Accordion Control 986
NoBot Control 988
PasswordStrength Control 990
Rating Control 991
TabContainer Control 993
Summary **994**

Chapter 21: Security **995**

Authentication and Authorization **996**
Applying Authentication Measures **996**
The <authentication> Node 997
Windows-Based Authentication 998
Forms-Based Authentication 1006
Passport Authentication 1016
Authenticating Specific Files and Folders **1016**
Programmatic Authorization **1017**
Working with User.Identity 1018
Working with User.IsInRole() 1019
Pulling More Information with WindowsIdentity 1020
Identity and Impersonation **1023**
Securing Through IIS **1025**
IP Address and Domain Name Restrictions 1025
Working with File Extensions 1027
Using the ASP.NET MMC Snap-In 1031
Using the IIS 7.0 Manager 1032
Summary **1032**

Chapter 22: State Management **1033**

What Are Your Choices? **1034**
Understanding the Session Object in ASP.NET **1036**
Sessions and the Event Model 1036
Configuring Session State Management 1038
In-Process Session State 1038
Out-of-Process Session State 1046
SQL-Backed Session State 1051
Extending Session State with Other Providers 1056
Cookieless Session State 1057
Choosing the Correct Way to Maintain State 1058
The Application Object **1059**
QueryStrings **1060**

Cookies **1060**
PostBacks and Cross-Page PostBacks **1061**
Hidden Fields, ViewState, and ControlState **1063**
Using HttpContext.Current.Items for Very Short-Term Storage **1067**
Summary **1069**

Chapter 23: Caching **1071**

Caching **1071**
Output Caching 1071
Partial Page (UserControl) Caching 1074
Post-Cache Substitution 1075
HttpCachePolicy and Client-Side Caching 1078
Caching Programmatically **1080**
Data Caching Using the Cache Object 1080
Controlling the ASP.NET Cache 1081
Cache Dependencies 1081
Using the SQL Server Cache Dependency **1087**
Enabling Databases for SQL Server Cache Invalidation 1088
Enabling Tables for SQL Server Cache Invalidation 1088
Looking at SQL Server 2000 1089
Looking at the Tables That Are Enabled 1090
Disabling a Table for SQL Server Cache Invalidation 1090
Disabling a Database for SQL Server Cache Invalidation 1091
SQL Server 2005 Cache Invalidation 1091
Configuring Your ASP.NET Application **1092**
Testing SQL Server Cache Invalidation **1094**
Adding More Than One Table to a Page 1096
Attaching SQL Server Cache Dependencies to the Request Object 1096
Attaching SQL Server Cache Dependencies to the Cache Object 1097
Summary **1101**

Chapter 24: Debugging and Error Handling **1103**

Design-Time Support **1103**
Syntax Notifications 1104
Immediate and Command Window 1106
Task List 1106
Tracing **1107**
System.Diagnostics.Trace and ASP.NET's Page.Trace 1108
Page-Level Tracing 1108
Application Tracing 1108
Viewing Trace Data 1109

Tracing from Components 1113
Trace Forwarding 1114
TraceListeners 1114
Diagnostic Switches 1119
Web Events 1121
Debugging 1122
What's Required 1123
IIS versus ASP.NET Development Server 1124
Starting a Debugging Session 1125
New Tools to Help You with Debugging 1128
Client-side Javascript Debugging 1131
SQL Stored Proc Debugging 1134
Exception and Error Handling 1134
Handling Exceptions on a Page 1135
Handling Application Exceptions 1136
Http Status Codes 1137
Summary 1138

Chapter 25: File I/O and Streams 1139

Working with Drives, Directories, and Files 1140
The DriveInfo Class 1140
The Directory and DirectoryInfo Classes 1143
File and FileInfo 1149
Working with Paths 1154
File and Directory Properties, Attributes, and Access Control Lists 1158
Reading and Writing Files 1166
Streams 1167
Readers and Writers 1171
Compressing Streams 1176
Working with Serial Ports 1181
Network Communications 1182
WebRequest and WebResponse 1183
Sending Mail 1189
Summary 1190

Chapter 26: User and Server Controls 1193

User Controls 1194
Creating User Controls 1194
Interacting with User Controls 1196
Loading User Controls Dynamically 1198

Server Controls **1203**
WebControl Project Setup 1204
Control Attributes 1209
Control Rendering 1210
Adding Tag Attributes 1214
Styling HTML 1217
Themes and Skins 1220
Adding Client-Side Features 1222
Detecting and Reacting to Browser Capabilities 1231
Using ViewState 1234
Raising PostBack Events 1238
Handling PostBack Data 1242
Composite Controls 1244
Templated Controls 1247
Creating Control Design-Time Experiences 1254
Summary **1273**

Chapter 27: Modules and Handlers **1275**

Processing HTTP Requests **1275**
IIS 5/6 and ASP.NET 1275
IIS 7 and ASP.NET 1276
ASP.NET Request Processing 1277
HttpModules **1278**
HttpHandlers 1289
Summary **1295**

Chapter 28: Using Business Objects **1297**

Using Business Objects in ASP.NET 3.5 **1297**
Creating Precompiled .NET Business Objects 1298
Using Precompiled Business Objects in Your ASP.NET Applications 1301
COM Interop: Using COM Within .NET **1302**
The Runtime Callable Wrapper 1303
Using COM Objects in ASP.NET Code 1304
Error Handling 1309
Deploying COM Components with .NET Applications 1312
Using .NET from Unmanaged Code **1314**
The COM-Callable Wrapper 1314
Using .NET Components Within COM Objects 1316
Early versus Late Binding 1320

Error Handling 1320
Deploying .NET Components with COM Applications 1322
Summary 1324

Chapter 29: Building and Consuming Services 1325

Communication Between Disparate Systems 1325
Building a Simple XML Web Service 1327
The WebService Page Directive 1328
Looking at the Base Web Service Class File 1329
Exposing Custom Datasets as SOAP 1330
The XML Web Service Interface 1333
Consuming a Simple XML Web Service 1336
Adding a Web Reference 1336
Invoking the Web Service from the Client Application 1338
Transport Protocols for Web Services 1341
HTTP-GET 1342
HTTP-POST 1344
SOAP 1345
Overloading WebMethods 1346
Caching Web Service Responses 1349
SOAP Headers 1350
Building a Web Service with SOAP Headers 1351
Consuming a Web Service Using SOAP Headers 1353
Requesting Web Services Using SOAP 1.2 1355
Consuming Web Services Asynchronously 1357
Windows Communication Foundation 1360
The Larger Move to SOA 1360
WCF Overview 1361
Building a WCF Service 1362
Building the WCF Consumer 1370
Adding a Service Reference 1370
Working with Data Contracts 1374
Namespaces 1379
Summary 1379

Chapter 30: Localization 1381

Cultures and Regions 1381
Understanding Culture Types 1382
The ASP.NET Threads 1383
Server-Side Culture Declarations 1386
Client-Side Culture Declarations 1387
Translating Values and Behaviors 1389

ASP.NET 3.5 Resource Files **1397**
  Making Use of Local Resources 1397
  Making Use of Global Resources 1403
Looking at the Resource Editor **1406**
Summary **1407**

Chapter 31: Configuration **1409**

Configuration Overview **1410**
  Server Configuration Files 1411
  Application Configuration File 1413
  How Configuration Settings Are Applied 1414
  Detecting Configuration File Changes 1415
  Configuration File Format 1415
Common Configuration Settings **1416**
  Connecting Strings 1416
  Configuring Session State 1417
  Compilation Configuration 1421
  Browser Capabilities 1423
  Custom Errors 1426
  Authentication 1427
  Anonymous Identity 1430
  Authorization 1430
  Locking-Down Configuration Settings 1433
  ASP.NET Page Configuration 1433
  Include Files 1435
  Configuring ASP.NET Runtime Settings 1436
  Configuring the ASP.NET Worker Process 1438
  Storing Application-Specific Settings 1440
  Programming Configuration Files 1441
  Protecting Configuration Settings 1448
  Editing Configuration Files 1452
Creating Custom Sections **1453**
  Using the NameValueFileSectionHandler Object 1454
  Using the DictionarySectionHandler Object 1456
  Using the SingleTagSectionHandler Object 1457
  Using Your Own Custom Configuration Handler 1458
Summary **1460**

Chapter 32: Instrumentation **1461**

Working with the Event Log **1461**
  Reading from the Event Log 1462
  Writing to the Event Logs 1464

Using Performance Counters **1468**
Viewing Performance Counters Through an Administration Tool 1468
Building a Browser-Based Administrative Tool 1470
Application Tracing **1476**
Understanding Health Monitoring **1477**
The Health Monitoring Provider Model 1477
Health Monitoring Configuration 1479
Writing Events via Configuration: Running the Example 1486
Routing Events to SQL Server 1487
Buffering Web Events 1490
E-mailing Web Events 1492
Summary **1498**

Chapter 33: Administration and Management **1499**

The ASP.NET Web Site Administration Tool **1499**
The Home Tab 1501
The Security Tab 1501
The Application Tab 1510
The Provider Tab 1512
Configuring ASP.NET in IIS on Vista **1514**
.NET Compilation 1517
.NET Globalization 1518
.NET Profile 1518
.NET Roles 1520
.NET Trust Levels 1520
.NET Users 1521
Application Settings 1522
Connection Strings 1523
Pages and Controls 1524
Providers 1524
Session State 1524
SMTP E-mail 1526
Summary **1527**

Chapter 34: Packaging and Deploying ASP.NET Applications **1529**

Deployment Pieces **1530**
Steps to Take before Deploying **1530**
Methods of Deploying Web Applications **1531**
Using XCopy 1531
Using the VS Copy Web Site Option 1534
Deploying a Precompiled Web Application 1537
Building an Installer Program 1539

Looking More Closely at Installer Options **1547**
Working with the Deployment Project Properties 1550
The File System Editor 1554
The Registry Editor 1557
The File Types Editor 1559
The User Interface Editor 1561
The Custom Actions Editor 1562
The Launch Conditions Editor 1564
Summary **1565**

Appendix A: Migrating Older ASP.NET Projects 1567

Migrating Is Not Difficult **1567**
Running Multiple Versions of the Framework Side by Side 1568
Upgrading Your ASP.NET Applications 1568
When Mixing Versions — Forms Authentication **1570**
Upgrading — ASP.NET Reserved Folders **1571**
ASP.NET 3.5 Pages Come as XHTML **1571**
No Hard-Coded .js Files in ASP.NET 3.5 **1573**
Converting ASP.NET 1.x Applications in Visual Studio 2008 **1574**
Migrating from ASP.NET 2.0 to 3.5 **1580**

Appendix B: ASP.NET Ultimate Tools 1583

Debugging Made Easier **1583**
Firebug 1584
YSlow 1585
IE WebDeveloper Toolbar and Firefox WebDeveloper 1586
Aptana Studio — Javascript IDE 1588
Profilers: dotTrace or ANTS 1589
References **1590**
PositionIsEverything.net, QuirksMode.org, and HTMLDog.com 1590
Visibone 1590
www.asp.net 1590
Tidying Up Your Code **1591**
Refactor! for ASP.NET from Devexpress 1591
Code Style Enforcer 1592
Packer for .NET — Javascript Minimizer 1593
Visual Studio Add-ins **1594**
ASPX Edit Helper Add-In for Visual Studio 1595
Power Toys Pack Installer 1596
Extending ASP.NET **1597**
ASP.NET AJAX Control Toolkit 1597

Atif Aziz's ELMAH — Error Logging Modules and Handlers 1598
Helicon's ISAPI_Rewrite 1599
General Purpose Developer Tools 1600
Telerik's Online Code Converter 1600
WinMerge and Differencing Tools 1601
Reflector 1602
CR_Documentor 1603
Process Explorer 1604
Summary 1605

Appendix C: Silverlight 1607

Extending ASP.NET Apps with Silverlight 1607
Step 1: A Basic ASP.NET Application 1609
Finding Vector-Based Content 1610
Converting Vector Content to XAML 1611
Tools for Viewing and Editing XAML 1614
Integrating with Your Existing ASP.NET Site 1620
Receiving Silverlight Events in JavaScript 1623
Accessing Silverlight Elements from JavaScript Events 1625
Summary 1626

Appendix D: ASP.NET Online Resources 1627

Author Blogs 1627
ASP.NET Influential Blogs 1627
Web Sites 1628

Index 1629

Introduction

Simply put, you will find that ASP.NET 3.5 is an amazing technology to use to build your Web solutions! When ASP.NET 1.0 was introduced in 2000, many considered it a revolutionary leap forward in the area of Web application development. ASP.NET 2.0 was just as exciting and revolutionary and ASP.NET 3.5 is continuing a forward march in providing the best framework today in building applications for the Web. Although the foundation of ASP.NET was laid with the release of ASP.NET 1.0, ASP.NET 3.5 continues to build on this foundation by focusing on the area of developer productivity.

This book covers the whole of ASP.NET. It not only introduces new topics, it also shows you examples of these new technologies in action. So sit back, pull up that keyboard, and let's have some fun!

A Little Bit of History

Before organizations were even thinking about developing applications for the Internet, much of the application development focused on thick desktop applications. These thick-client applications were used for everything from home computing and gaming to office productivity and more. No end was in sight for the popularity of this application model.

During that time, Microsoft developers developed its thick-client applications using mainly Visual Basic (VB).

Visual Basic was not only a programming language; it was tied to an IDE that allowed for easy thick-client application development. In the Visual Basic model, developers could drop controls onto a form, set properties for these controls, and provide code behind them to manipulate the events of the control. For example, when an end user clicked a button on one of the Visual Basic forms, the code behind the form handled the event.

Then, in the mid-1990s, the Internet arrived on the scene. Microsoft was unable to move the Visual Basic model to the development of Internet-based applications. The Internet definitely had a lot of power, and right away, the problems facing the thick-client application model were revealed. Internet-based applications created a single instance of the application that everyone could access. Having one instance of an application meant that when the application was upgraded or patched, the changes made to this single instance were immediately available to each and every user visiting the application through a browser.

To participate in the Web application world, Microsoft developed Active Server Pages (ASP). ASP was a quick and easy way to develop Web pages. ASP pages consisted of a single page that contained a mix of markup and languages. The power of ASP was that you could include VBScript or JScript code instructions in the page executed on the Web server before the page was sent to the end user's Web browser. This was an easy way to create dynamic Web pages customized based on instructions dictated by the developer.

ASP used script between brackets and percentage signs — `<% %>` — to control server-side behaviors. A developer could then build an ASP page by starting with a set of static HTML. Any dynamic element needed by the page was defined using a scripting language (such as VBScript or JScript). When a user requested the page from the server by using a browser, the `asp.dll` (an ISAPI application that provided a bridge between the scripting language and the Web server) would take hold of the page and define all the dynamic aspects of the page on-the-fly based on the programming logic specified in the script. After all the dynamic aspects of the page were defined, the result was an HTML page output to the browser of the requesting client.

As the Web application model developed, more and more languages mixed in with the static HTML to help manipulate the behavior and look of the output page. Over time, such a large number of languages, scripts, and plain text could be placed in a typical ASP page that developers began to refer to pages that utilized these features as *spaghetti code*. For example, it was quite possible to have a page that used HTML, VBScript, JavaScript, Cascading Style Sheets, T-SQL, and more. In certain instances, it became a manageability nightmare.

ASP evolved and new versions were released. ASP 2.0 and 3.0 were popular because the technology made it relatively straightforward and easy to create Web pages. Their popularity was enhanced because they appeared in the late 1990s, just as the dotcom era was born. During this time, a mountain of new Web pages and portals were developed, and ASP was one of the leading technologies individuals and companies used to build them. Even today, you can still find a lot of `.asp` pages on the Internet — including some of Microsoft's own Web pages.

However, even at the time of the final release of Active Server Pages in late 1998, Microsoft employees Marc Anders and Scott Guthrie had other ideas. Their ideas generated what they called XSP (an abbreviation with no meaning) — a new way of creating Web applications in an object-oriented manner instead of the procedural manner of ASP 3.0. They showed their idea to many different groups within Microsoft, and they were well received. In the summer of 2000, the beta of what was then called ASP+ was released at Microsoft's Professional Developers Conference. The attendees eagerly started working with it. When the technology became available (with the final release of the .NET Framework 1.0), it was renamed ASP.NET — receiving the .NET moniker that most of Microsoft's new products were receiving at that time.

Before the introduction of .NET, the model that classic ASP provided and what developed in Visual Basic were so different that few VB developers also developed Web applications-and few Web application developers also developed the thick-client applications of the VB world. There was a great divide. ASP.NET bridged this gap. ASP.NET brought a Visual Basic–style eventing model to Web application development, providing much-needed state management techniques over stateless HTTP. Its model is much like the earlier Visual Basic model in that a developer can drag and drop a control onto a design surface or form, manipulate the control's properties, and even work with the code behind these controls to act on certain events that occur during their lifecycles. What ASP.NET created is really the best of both models, as you will see throughout this book.

I know you will enjoy working with this latest release of ASP.NET 3.5. Nothing is better than getting your hands on a new technology and seeing what is possible. The following section discusses the goals of ASP.NET so you can find out what to expect from this new offering!

The Goals of ASP.NET

ASP.NET 3.5 is another major release of the product and builds upon the core .NET Framework 2.0 with additional classes and capabilities. This release of the Framework was code-named *Orcas* internally at Microsoft. You might hear others referring to this release of ASP.NET as *ASP.NET Orcas*. ASP.NET 3.5 continues on a path to make ASP.NET developers the most productive developers in the Web space.

Ever since the release of ASP.NET 2.0, the Microsoft team has had goals focused around developer productivity, administration, and management, as well as performance and scalability.

Developer Productivity

Much of the focus of ASP.NET 3.5 is on productivity. Huge productivity gains were made with the release of ASP.NET 1.*x*; could it be possible to expand further on those gains?

One goal the development team had for ASP.NET was to eliminate much of the tedious coding that ASP.NET originally required and to make common ASP.NET tasks easier. The developer productivity capabilities are presented throughout this book. Before venturing into these capabilities, this introduction will first start by taking a look at the older ASP.NET 1.0 technology in order to make a comparison to ASP.NET 3.5. Listing I-1 provides an example of using ASP.NET 1.0 to build a table in a Web page that includes the capability to perform simple paging of the data provided.

Listing I-1: Showing data in a DataGrid server control with paging enabled (VB only)

```
<%@ Page Language="VB" AutoEventWireup="True" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<script runat="server">

    Private Sub Page_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs)
        If Not Page.IsPostBack Then
            BindData()
        End If
    End Sub

    Private Sub BindData()
        Dim conn As SqlConnection = New SqlConnection("server='localhost';
           trusted_connection=true; Database='Northwind'")
        Dim cmd As SqlCommand = New SqlCommand("Select * From Customers", conn)
        conn.Open()

        Dim da As SqlDataAdapter = New SqlDataAdapter(cmd)
        Dim ds As New DataSet

        da.Fill(ds, "Customers")

        DataGrid1.DataSource = ds
        DataGrid1.DataBind()
    End Sub

    Private Sub DataGrid1_PageIndexChanged(ByVal source As Object, _
      ByVal e As System.Web.UI.WebControls.DataGridPageChangedEventArgs)
        DataGrid1.CurrentPageIndex = e.NewPageIndex
        BindData()
    End Sub

</script>
<html>
```

```
<head>
</head>
<body>
    <form runat="server">
        <asp:DataGrid id="DataGrid1" runat="server" AllowPaging="True"
         OnPageIndexChanged="DataGrid1_PageIndexChanged"></asp:DataGrid>
    </form>
</body>
</html>
```

Although quite a bit of code is used here, this is a dramatic improvement over the amount of code required to accomplish this task using classic Active Server Pages 3.0. We will not go into the details of this older code; we just want to demonstrate that in order to add any additional common functionality (such as paging) for the data shown in a table, the developer had to create custom code.

This is one area where the developer productivity gains are most evident. ASP.NET 3.5 provides a control called the GridView server control. This control is much like the DataGrid server control, but the GridView server control (besides offering many other additional features) contains the built-in capability to apply paging, sorting, and editing of data with relatively little work on your part. Listing I-2 shows you an example of the GridView server control. This example builds a table of data from the Customers table in the Northwind database that includes paging.

Listing I-2: Viewing a paged dataset with the new GridView server control

```
<%@ Page Language="VB" %>

<script runat="server">

</script>

<html xmlns=http://www.w3.org/1999/xhtml>
<head runat="server">
    <title>GridView Demo</title>
</head>
<body>
    <form runat="server">
        <asp:GridView ID="GridView1" Runat="server" AllowPaging="True"
         DataSourceId="Sqldatasource1" />
        <asp:SqlDataSource ID="SqlDataSource1" Runat="server"
         SelectCommand="Select * From Customers"
         ProviderName="System.Data.OleDb"
         ConnectionString="Provider=SQLOLEDB;Server=localhost;uid=sa;
         pwd=password;database=Northwind" />
    </form>
</body>
</html>
```

That’s it! You can apply paging by using a couple of new server controls. You turn on this capability using a server control attribute, the `AllowPaging` attribute of the GridView control:

```
<asp:GridView ID="GridView1" Runat="server" AllowPaging="True"
 DataSourceId="SqlDataSource1" />
```

The other interesting event occurs in the code section of the document:

```
<script runat="server">

</script>
```

These two lines of code are not actually needed to run the file. They are included here to make a point — *you don't need to write any server-side code to make this all work!* You have to include only some server controls: one control to get the data and one control to display the data. Then the controls are wired together.

Performance and Scalability

One of the highlights for ASP.NET that was set by the Microsoft team was to provide the world's fastest Web application server. This book also addresses a number of performance tactics available in ASP.NET 3.5.

One of the most exciting performance capabilities is the caching capability aimed at exploiting Microsoft's SQL Server. ASP.NET 3.5 includes a feature called *SQL cache invalidation*. Before ASP.NET 2.0, it was possible to cache the results that came from SQL Server and to update the cache based on a time interval — for example, every 15 seconds or so. This meant that the end user might see stale data if the result set changed sometime during that 15-second period.

In some cases, this time interval result set is unacceptable. In an ideal situation, the result set stored in the cache is destroyed if any underlying change occurs in the source from which the result set is retrieve — in this case, SQL Server. With ASP.NET 3.5, you can make this happen with the use of SQL cache invalidation. This means that when the result set from SQL Server changes, the output cache is triggered to change, and the end user always sees the latest result set. The data presented is never stale.

ASP.NET 3.5 provides 64-bit support. This means that you can run your ASP.NET applications on 64-bit Intel or AMD processors.

Because ASP.NET 3.5 is fully backward compatible with ASP.NET 1.0, 1.1 and 2.0, you can now take any former ASP.NET application, recompile the application on the .NET Framework 3.5, and run it on a 64-bit processor.

Additional Features of ASP.NET 3.5

You just learned some of the main goals of the ASP.NET team that built ASP.NET. To achieve these goals, the team built a mountain of features into each and every release of ASP.NET. A few of these features are described in the following sections.

New Developer Infrastructures

An exciting aspect of ASP.NET 3.5 is that there are infrastructures are in place for you to use in your applications. The ASP.NET team selected some of the most common programming operations performed with Web applications to be built directly into ASP.NET. This saves you considerable time and coding.

Membership and Role Management

Prior to ASP.NET 2.0, if you were developing a portal that required users to log in to the application to gain privileged access, invariably you had to create it yourself. It can be tricky to create applications with areas that are accessible only to select individuals.

You will find with ASP.NET 3.5, this capability is built in. You can validate users as shown in Listing I-3.

Listing I-3: Validating a user in code

VB

```
If (Membership.ValidateUser (Username.Text, Password.Text)) Then
   ' Allow access code here
End If
```

C#

```
if (Membership.ValidateUser (Username.Text, Password.Text)) {
   // Allow access code here
}
```

A series of APIs, controls, and providers in ASP.NET 3.5 enable you to control an application's user membership and role management. Using these APIs, you can easily manage users and their complex roles — creating, deleting, and editing them. You get all this capability by using the APIs or a built-in Web tool called the Web Site Administration Tool.

As far as storing users and their roles, ASP.NET 3.5 uses an `.mdf` file (the file type for the SQL Server Express Edition) for storing all users and roles. You are in no way limited to just this data store, however. You can expand everything offered to you by ASP.NET and build your own providers using whatever you fancy as a data store. For example, if you want to build your user store in LDAP or within an Oracle database, you can do so quite easily.

Personalization

One advanced feature that portals love to offer their membership base is the capability to personalize their offerings so that end users can make the site look and function however they want. The capability to personalize an application and store the personalization settings is completely built into the ASP.NET Framework.

Because personalization usually revolves around a user and possibly a role that this user participates in, the personalization architecture can be closely tied to the membership and role infrastructures. You have a couple of options for storing the created personalization settings. The capability to store these settings in either Microsoft Access or in SQL Server is built into ASP.NET 3.5. As with the capabilities of the membership and role APIs, you can use the flexible provider model, and then either change how the built-in provider uses the available data store or build your own custom data provider to work with a completely new data store. The personalization API also supports a union of data stores, meaning that you can use more than one data store if you want.

Because it is so easy to create a site for customization using these new APIs, this feature is quite a value-add for any application you build.

The ASP.NET Portal Framework

During the days of ASP.NET 1.0, developers could go to the ASP.NET team's site (found at `asp.net`) and download some Web application demos such as IBuySpy. These demos are known as Developer Solution

Kits and are used as the basis for many of the Web sites on the Internet today. Some were even extended into opensource frameworks such as DotNetNuke.

The nice thing about IBuySpy was that you could use the code it provided as a basis to build either a Web store or a portal. You simply took the base code as a starting point and extended it. For example, you could change the look and feel of the presentation part of the code or introduce advanced functionality into its modular architecture. Developer Solution Kits are quite popular because they make performing these types of operations so easy.

Because of the popularity of frameworks such as IBuySpy, ASP.NET 3.5 offers built-in capability for using Web Parts to easily build portals. The possibilities for what you can build using the Portal Framework is astounding. The power of building using Web Parts is that it easily enables end users to completely customize the portal for their own preferences.

Site Navigation

The ASP.NET team members realize that end users want to navigate through applications with ease. The mechanics to make this work in a logical manner are sometimes hard to code. The team solved the problem in ASP.NET with a series of navigation-based server controls.

First, you can build a site map for your application in an XML file that specific controls can inherently work from. Listing I-4 shows a sample site map file.

Listing I-4: An example of a site map file

```
<?xml version="1.0" encoding="utf-8" ?>

<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
   <siteMapNode title="Home" description="Home Page" url="default.aspx">
      <siteMapNode title="News" description="The Latest News" url="News.aspx">
         <siteMapNode title="U.S." description="U.S. News"
          url="News.aspx?cat=us" />
         <siteMapNode title="World" description="World News"
          url="News.aspx?cat=world" />
         <siteMapNode title="Technology" description="Technology News"
          url="News.aspx?cat=tech" />
         <siteMapNode title="Sports" description="Sports News"
          url="News.aspx?cat=sport" />
      </siteMapNode>
      <siteMapNode title="Finance" description="The Latest Financial Information"
        url="Finance.aspx">
         <siteMapNode title="Quotes" description="Get the Latest Quotes"
          url="Quotes.aspx" />
         <siteMapNode title="Markets" description="The Latest Market Information"
          url="Markets.aspx">
            <siteMapNode title="U.S. Market Report"
             description="Looking at the U.S. Market" url="MarketsUS.aspx" />
            <siteMapNode title="NYSE"
             description="The New York Stock Exchange" url="NYSE.aspx" />
         </siteMapNode>
         <siteMapNode title="Funds" description="Mutual Funds"
          url="Funds.aspx" />
      </siteMapNode>
      <siteMapNode title="Weather" description="The Latest Weather"
       url="Weather.aspx" />
   </siteMapNode>
</siteMap>
```

After you have a site map in place, you can use this file as the data source behind a couple of site navigation server controls, such as the TreeView and the SiteMapPath server controls. The TreeView server control enables you to place an expandable site navigation system in your application. Figure I-1 shows you an example of one of the many looks you can give the TreeView server control.

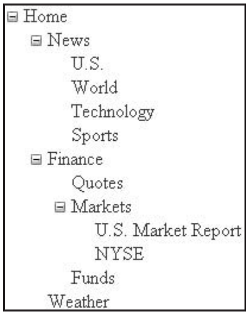


Figure I-1

SiteMapPath is a control that provides the capability to place what some call *breadcrumb navigation* in your application so that the end user can see the path that he has taken in the application and can easily navigate to higher levels in the tree. Figure I-2 shows you an example of the SiteMapPath server control at work.

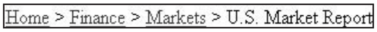


Figure I-2

These site navigation capabilities provide a great way to get programmatic access to the site layout and even to take into account things like end-user roles to determine which parts of the site to show.

The ASP.NET Compilation System

Compilation in ASP.NET 1.0 was always a tricky scenario. With ASP.NET 1.0, you could build an application's code-behind files using ASP.NET and Visual Studio, deploy it, and then watch as the `.aspx` files were compiled page by page as each page was requested. If you made any changes to the code-behind file in ASP.NET 1.0, it was not reflected in your application until the entire application was rebuilt. That meant that the same page-by-page request had to be done again before the entire application was recompiled.

Everything about how ASP.NET 1.0 worked with classes and compilation is different from how it is in ASP.NET 3.5. The mechanics of the compilation system actually begin with how a page is structured in ASP.NET 3.5. In ASP.NET 1.0, either you constructed your pages using the code-behind model or by placing all the server code inline between `<script>` tags on your `.aspx` page. Most pages were constructed using the code-behind model because this was the default when using Visual Studio .NET 2002 or 2003. It was quite difficult to create your page using the inline style in these IDEs. If you did, you were deprived of the use of IntelliSense, which can be quite the lifesaver when working with the tremendously large collection of classes that the .NET Framework offers.

ASP.NET 3.5 offers a different code-behind model than the 1.0/1.1 days because the .NET Framework 3.5 has the capability to work with *partial classes* (also called partial types). Upon compilation, the separate files are combined into a single offering. This gives you much cleaner code-behind pages. The code that was part of the `Web Form Designer Generated` section of your classes is separated from the code-behind classes that you create yourself. Contrast this with the ASP.NET 1.0 `.aspx` file's need to derive from its own code-behind file to represent a single logical page.

ASP.NET 3.5 applications can include a `\App_Code` directory where you place your class's source. Any class placed here is dynamically compiled and reflected in the application. You do not use a separate build process when you make changes as you did with ASP.NET 1.0. This is a *just save and hit* deployment model like the one in classic ASP 3.0. Visual Studio 2008 also automatically provides IntelliSense for any objects that are placed in the `\App_Code` directory, whether you are working with the code-behind model or are coding inline.

ASP.NET 3.5 also provides you with tools that enable you to precompile your ASP.NET applications — both `.aspx` pages and code behind — so that no page within your application has latency when it is retrieved for the first time. Doing this is also a great way to discover any errors in the pages without invoking every page. Precompiling your ASP.NET 2.0 applications is as simple as using `aspnet_compiler.exe` and employing some of the available flags. As you precompile your entire application, you also receive error notifications if any errors are found anywhere within it. Precompilation also enables you to deliver only the created assembly to the deployment server, thereby protecting your code from snooping, unwanted changes, and tampering after deployment. You see examples of these scenarios later in this book.

Health Monitoring for Your ASP.NET Applications

The built-in health monitoring capabilities are rather significant features designed to make it easier to manage a deployed ASP.NET application. Health monitoring provides what the term implies — the capability to monitor the health and performance of your deployed ASP.NET applications.

ASP.NET health monitoring is built around various health monitoring events (which are referred to as *Web events*) occurring in your application. Using the health monitoring system enables you to perform event logging for Web events such as failed logins, application starts and stops, or any unhandled exceptions. The event logging can occur in more than one place; therefore, you can log to the event log or even back to a database. In addition to performing this disk-based logging, you can also use the system to e-mail health-monitoring information.

Besides working with specific events in your application, you can also use the health monitoring system to take health snapshots of a running application. As you can with most systems that are built into ASP.NET 3.5, you are able to extend the health monitoring system and create your own events for recording application information.

Health monitoring is already enabled by default in the system `.config` files. The default setup for health monitoring logs all errors and failure audits to the event log. For instance, throwing an error in your application results in an error notification in the Application log.

You can change the default event logging behaviors simply by making some minor changes to your application's `web.config` file. For instance, suppose that you want to store this error event information in a SQL Express file contained within the application. This change can be made by adding a `<healthMonitoring>` node to your `web.config` file as presented in Listing I-5.

Listing I-5: Defining health monitoring in the web.config file

```
<healthMonitoring enabled="true">
   <providers>
      <clear />
      <add name="SqlWebEventProvider" connectionStringName="LocalSqlServer"
       maxEventDetailsLength="1073741823" buffer="false" bufferMode="Notification"
       type="System.Web.Management.SqlWebEventProvider,
          System.Web,Version=2.0.0.0,Culture=neutral,
          PublicKeyToken=b03f5f7f11d50a3a"/>
   </providers>
   <rules>
      <clear />
      <add name="All Errors Default" eventName="All Errors"
       provider="SqlWebEventProvider"
       profile="Default" minInstances="1" maxLimit="Infinite"
       minInterval="00:01:00" custom="" />
      <add name="Failure Audits Default" eventName="Failure Audits"
       provider="SqlWebEventProvider" profile="Default" minInstances="1"
       maxLimit="Infinite" minInterval="00:01:00" custom="" />
   </rules>
</healthMonitoring>
```

After this change, events are logged in the `ASPNETDB.MDF` file that is automatically created on your behalf if it does not already exist in your project.

Opening up this SQL Express file, you will find an `aspnet_WebEvent_Events` table where all this information is stored.

You will learn much more about the health monitoring capabilities provided with ASP.NET 3.5 in Chapter 32.

Reading and Writing Configuration Settings

Using the `WebConfigurationManager` class, you have the capability to read and write to the server or application configuration files. This means that you can write and read settings in the `machine.config` or the `web.config` files that your application uses.

The capability to read and write to configuration files is not limited to working with the local machine in which your application resides. You can also perform these operations on remote servers and applications.

Of course, a GUI-way exists in which you can perform these read or change operations on the configuration files at your disposal. The exciting thing, however, is that the built-in GUI tools that provide this functionality (such as the ASP.NET MMC snap-in when using Windows XP, or the new IIS interface if you are using Windows Vista) use the `WebConfigurationManager` class that is also available for building custom administration tools.

Listing I-6 shows an example of reading a connection string from an application's `Web.config` file.

Listing I-6: Reading a connection string from the application's Web.config file

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   Try
      Dim connectionString As String = _
         ConfigurationManager.ConnectionStrings("Northwind").
            ConnectionString.ToString()
      Label1.Text = connectionString
   Catch ex As Exception
      Label1.Text = "No connection string found."
   End Try
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
   try
   {
      string connectionString =
         ConfigurationManager.ConnectionStrings["Northwind"].
            ConnectionString.ToString();
      Label1.Text = connectionString;
   }
   catch (Exception)
   {
      Label1.Text = "No connection string found.";
   }
}
```

This little bit of code writes the Northwind connection string found in the `web.config` file to the screen using a Label control. As you can see, it is rather simple to grab items from the configuration file.

Localization

ASP.NET is making it easier to localize applications than ever before. In addition to using Visual Studio, you can create resource files (`.resx`) that allow you to dynamically change the pages you create based upon the culture settings of the requestor.

ASP.NET 3.5 provides the capability to provide resources application-wide or just to particular pages in your application through the use of two new application folders — `App_GlobalResources` and `App_LocalResources`.

The items defined in any `.resx` files you create are then accessible directly in the ASP.NET server controls or programmatically using expressions such as:

```
<%= Resources.Resource.Question %>
```

This system is straightforward and simple to implement. This topic is covered in greater detail in Chapter 30

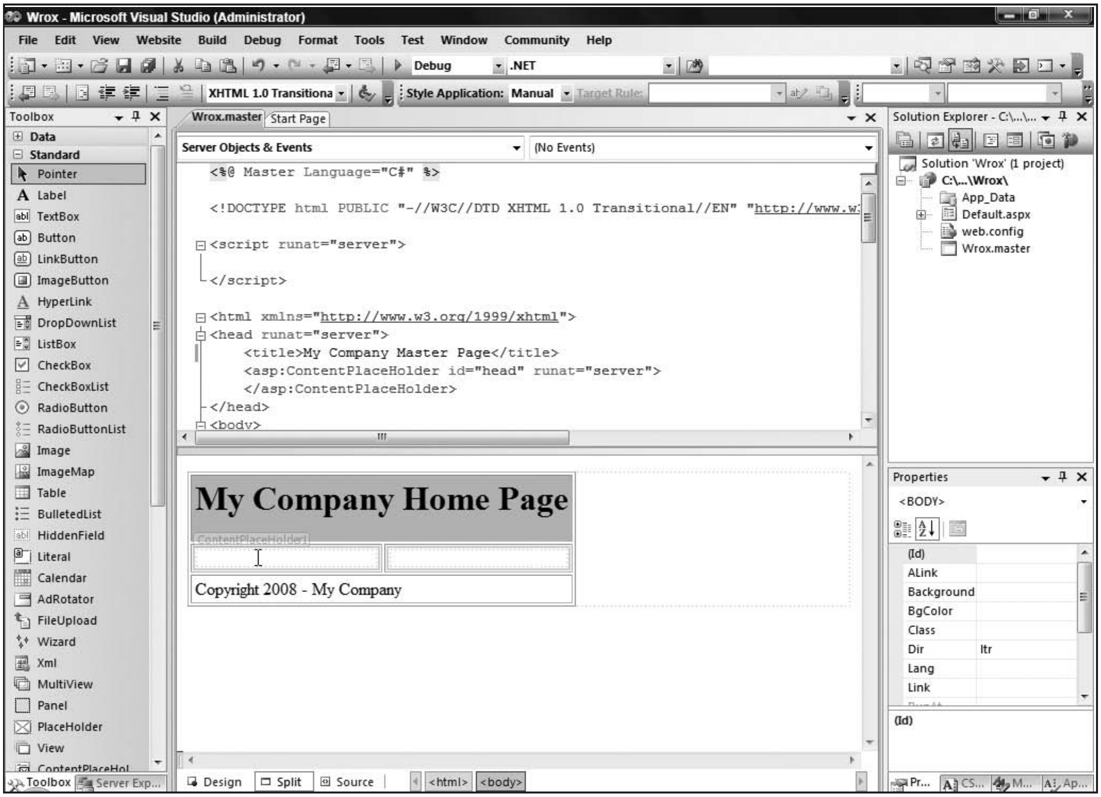
Expanding on the Page Framework

ASP.NET pages can be built based upon visual inheritance. This was possible in the Windows Forms world, but it is something that is relatively new with ASP.NET. You also gain the capability to easily apply a consistent look and feel to the pages of your application by using themes. Many of the difficulties in working with ADO.NET is made easier through a series of data source controls that take care of accessing and retrieving data from a large collection of data stores.

Master Pages

With the capability of *master pages* in ASP.NET, you can use visual inheritance within your ASP.NET applications. Because many ASP.NET applications have a similar structure throughout their pages, it is logical to build a page template once and use that same template throughout the application.

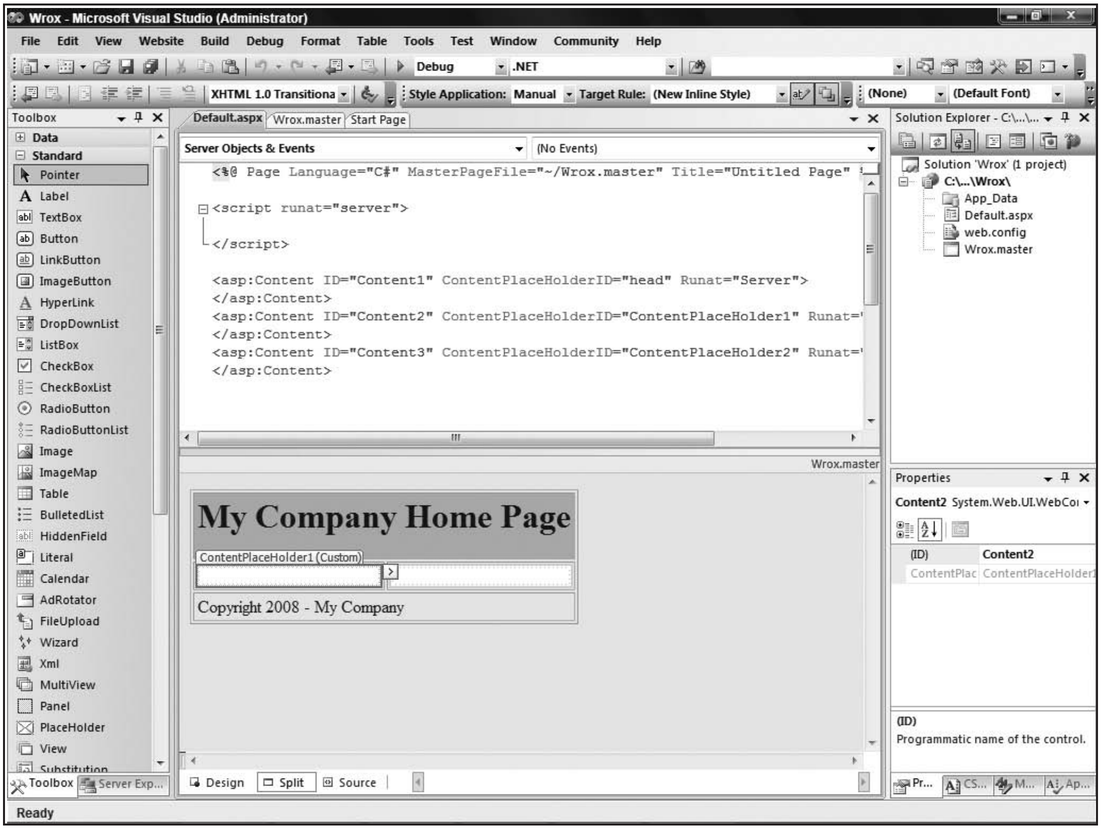
In ASP.NET, you do this by creating a `.master` page, as shown in Figure I-3.


Wrox - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Format Tools Test Window Community Help
Debug
.NET
XHTML 1.0 Transitiona
Style Application: Manual
Target Rule:
Toolbox
Data
Standard
Pointer
Label
TextBox
Button
LinkButton
ImageButton
HyperLink
DropDownList
ListBox
CheckBox
CheckBoxList
RadioButton
RadioButtonList
Image
ImageMap
Table
BulletedList
HiddenField
Literal
Calendar
AdRotator
FileUpload
Wizard
Xml
MultiView
Panel
PlaceHolder
View
ContentPlaceHol
Toolbox
Server Exp...
Wrox.master
Start Page
Server Objects & Events
(No Events)
My Company Home Page
ContentPlaceHolder1
Copyright 2008 - My Company
Design
Split
Source
<html>
<body>
Solution Explorer - C:\...\...
Solution 'Wrox' (1 project)
C:\...\Wrox\
App_Data
Default.aspx
web.config
Wrox.master
Properties
<BODY>
(Id)
ALink
Background
BgColor
Class
Dir
ltr
Lang
Link
(Id)
Pr...
CS...
M...
Ap...

```
<%@ Master Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>My Company Master Page</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
```

Figure I-3

An example master page might include a header, footer, and any other elements that all the pages can share. Besides these core elements, which you might want on every page that inherits and uses this template, you can place `<asp:ContentPlaceHolder>` server controls within the master page itself for the subpages (or content pages) to use in order to change specific regions of the master page template. The editing of the subpage is shown in Figure I-4.


Wrox - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Format Table Tools Test Window Community Help
Debug
.NET
XHTML 1.0 Transitiona
Style Application: Manual
Target Rule: (New Inline Style)
(None)
(Default Font)
Toolbox
Data
Standard
Pointer
Label
TextBox
Button
LinkButton
ImageButton
HyperLink
DropDownList
ListBox
CheckBox
CheckBoxList
RadioButton
RadioButtonList
Image
ImageMap
Table
BulletedList
HiddenField
Literal
Calendar
AdRotator
FileUpload
Wizard
Xml
MultiView
Panel
PlaceHolder
View
Substitution
Toolbox
Server Exp...
Default.aspx
Wrox.master
Start Page
Server Objects & Events
(No Events)
Wrox.master
My Company Home Page
ContentPlaceHolder1 (Custom)
Copyright 2008 - My Company
Design
Split
Source
Solution Explorer - C:\...\...
Solution 'Wrox' (1 project)
C:\...\Wrox\
App_Data
Default.aspx
web.config
Wrox.master
Properties
Content2 System.Web.UI.WebCo
(ID)
Content2
ContentPlac
ContentPlaceHolder
(ID)
Programmatic name of the control.
Pr...
CS...
M...
Ap...
Ready

```
<%@ Page Language="C#" MasterPageFile="~/Wrox.master" Title="Untitled Page"

<script runat="server">

</script>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat="
</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="ContentPlaceHolder2" Runat="
</asp:Content>
```

Figure I-4

When an end user invokes one of the subpages, he is actually looking at a single page compiled from both the subpage and the master page that the particular subpage inherited from. This also means that the server and client code from both pages are enabled on the new single page.

The nice thing about master pages is that you have a single place to make any changes that affect the entire site. This eliminates making changes to each and every page within an application.

Themes

The inclusion of themes in ASP.NET has made it quite simple to provide a consistent look and feel across your entire site. Themes are simple text files where you define the appearance of server controls that can be applied across the site, to a single page, or to a specific server control. You can also easily incorporate graphics and Cascading Style Sheets, in addition to server control definitions.

Themes are stored in the `/App_Theme` directory within the application root for use within that particular application. One cool capability of themes is that you can dynamically apply them based on settings that use the personalization service provided by ASP.NET. Each unique user of your portal or application can have her own personalized look and feel that she has chosen from your offerings.

Objects for Accessing Data

One of the more code-intensive tasks in ASP.NET 1.0 was the retrieval of data. In many cases, this meant working with a number of objects. If you have been working with ASP.NET for a while, you know that it was an involved process to display data from a Microsoft SQL Server table within a DataGrid server control. For instance, you first had to create a number of new objects. They included a `SqlConnection` object followed by a `SqlCommand` object. When those objects were in place, you then created a `SqlDataReader` to populate your DataGrid by binding the result to the DataGrid. In the end, a table appeared containing the contents of the data you were retrieving (such as the Customers table from the Northwind database).

ASP.NET today eliminates this intensive procedure with the introduction of a set of objects that work specifically with data access and retrieval. These new data controls are so easy to use that you access and retrieve data to populate your ASP.NET server controls without writing any code. You saw an example of this in Listing I-2, where an `<asp:SqlDataSource>` server control retrieved rows of data from the Customers table in the Northwind database from SQL Server. This SqlDataSource server control was then bound to the new GridView server control via the use of simple attributes within the GridView control itself. It really could not be any easier!

The great news about this functionality is that it is not limited to just Microsoft's SQL Server. In fact, several data source server controls are at your disposal. You also have the capability to create your own. In addition to the SqlDataSource server control, ASP.NET 3.5 includes the AccessDataSource, XmlDataSource, ObjectDataSource, SiteMapDataSource, and the new LinqDataSource server control. You use all these data controls later in this book.

What You Need for ASP.NET 3.5

You might find it best to install Visual Studio 2008 to work through the examples in this book; you can, however, just use Microsoft's Notepad and the command-line compilers that come with the .NET Framework 3.5. To work through *every* example in this book, you need the following:

- ☐ Windows Server 2003, Windows Server 2008, Windows 2000, Windows XP, or Windows Vista
- ☐ Visual Studio 2008 (this will install the .NET Framework 3.5)
- ☐ SQL Server 2000, 2005, or 2008
- ☐ Microsoft Access or SQL Server Express Edition

The nice thing is that you are not required to have Microsoft Internet Information Services (IIS) to work with ASP.NET 3.5 because ASP.NET includes a built-in Web server based on the previously released Microsoft Cassini technology. Moreover, if you do not have a full blown version of SQL Server, don't be alarmed. Many examples that use this database can be altered to work with Microsoft's SQL Server Express Edition, which you will find free on the Internet.

Who Should Read This Book?

This book was written to introduce you to the features and capabilities that ASP.NET 3.5 offers, as well as to give you an explanation of the foundation that ASP.NET provides. We assume you have a general understanding of Web technologies, such as previous versions of ASP.NET, Active Server Pages 2.0/3.0, or JavaServer Pages. If you understand the basics of Web programming, you should not have much trouble following along with this book's content.

If you are brand new to ASP.NET, be sure to check out *Beginning ASP.NET 3.5: In C# and VB* by Imar Spaanjaars (Wiley Publishing, Inc., 2008) to help you understand the basics.

In addition to working with Web technologies, we also assume that you understand basic programming constructs, such as variables, `For Each` loops, and object-oriented programming.

You may also be wondering whether this book is for the Visual Basic developer or the C# developer. We are happy to say that it is for both! When the code differs substantially, this book provides examples in both VB and C#.

What This Book Covers

This book spends its time reviewing the 3.5 release of ASP.NET. Each major new feature included in ASP.NET 3.5 is covered in detail. The following list tells you something about the content of each chapter.

- ❑ **Chapter 1, ''Application and Page Frameworks.'':** The first chapter covers the frameworks of ASP.NET applications as well as the structure and frameworks provided for single ASP.NET pages. This chapter shows you how to build ASP.NET applications using IIS or the built-in Web server that comes with Visual Studio 2008. This chapter also shows you the folders and files that are part of ASP.NET. It discusses ways to compile code and shows you how to perform cross-page posting. This chapter ends by showing you easy ways to deal with your classes from within Visual Studio 2008.
- ❑ **Chapters 2, 3, and 4:** These three chapters are grouped here because they all deal with server controls. This batch of chapters starts by examining the idea of the server control and its pivotal role in ASP.NET development. In addition to looking at the server control framework, these chapters delve into the plethora of server controls that are at your disposal for ASP.NET development projects. Chapter 2, ''ASP.NET Server Controls and Client-Side Scripts,'' looks at the basics of working with server controls. Chapter 3, ''ASP.NET Web Server Controls,'' covers the controls that have been part of the ASP.NET technology since its initial release and the controls that have been added in each of the ASP.NET releases. Chapter 4, ''Validation Server Controls,'' describes a special group of server controls: those for validation. You can use these controls to create beginning-to-advanced form validations.
- ❑ **Chapter 5, ''Working with Master Pages.'':** Master pages are a great capability found in ASP.NET. They provide a means of creating templated pages that enable you to work with the entire application, as opposed to single pages. This chapter examines the creation of these templates and how to apply them to your content pages throughout an ASP.NET application.
- ❑ **Chapter 6, ''Themes and Skins.'':** The Cascading Style Sheet files you are allowed to use in ASP.NET 1.0/1.1 are simply not adequate in many regards, especially in the area of server controls. When using these early versions, the developer can never be sure of the HTML output these files might generate. This chapter looks at how to deal with the styles that your applications require and shows you how to create a centrally managed look-and-feel for all the pages of your application by using themes and the skin files that are part of a theme.
- ❑ **Chapters 7, ''Data Binding in ASP.NET 3.5.'':** One of the more important tasks of ASP.NET is presenting data, and this chapter shows you how to do that. ASP.NET provides a number of

controls to which you can attach data and present it to the end user. This chapter looks at the underlying capabilities that enable you to work with the data programmatically before issuing the data to a control.

- **Chapter 8, “Data Management with ADO.NET.”:** This chapter presents the ADO.NET data model provided by ASP.NET, which allows you to handle the retrieval, updating, and deleting of data quickly and logically. This new data model enables you to use one or two lines of code to get at data stored in everything from SQL Server to XML files.
- **Chapter 9, “Querying with LINQ.”:** The new addition to the .NET Framework 3.5 is the much-anticipated LINQ. LINQ is a set of extensions to the .NET Framework that encompass language-integrated query, set, and transform operations. This chapter introduces you to LINQ and how to effectively use this new feature in their web applications today.
- **Chapter 10, “Working with XML and LINQ to XML.”:** Without a doubt, XML has become one of the leading technologies used for data representation. For this reason, the .NET Framework and ASP.NET 3.5 have many capabilities built into their frameworks that enable you to easily extract, create, manipulate, and store XML. This chapter takes a close look at the XML technologies built into ASP.NET and the underlying .NET Framework.
- **Chapter 11, “IIS 7.0 Development.”:** Probably the most substantial release of IIS in its history, IIS 7.0 will change the way you host and work with your ASP.NET applications. IIS 7.0 is part of Windows Server 2008.
- **Chapter 17, “Introduction to the Provider Model.”:** A number of systems are built into ASP.NET that make the lives of developers so much easier and more productive than ever before. These systems are built upon an architecture called a *provider model*, which is rather extensible. This chapter gives an overview of this provider model and how it is used throughout ASP.NET 3.5.
- **Chapter 13, “Extending the Provider Model.”:** After an introduction of the provider model, this chapter looks at some of the ways to extend the provider model found in ASP.NET 3.5. This chapter also reviews a couple of sample extensions to the provider model.
- **Chapter 14, “Site Navigation.”:** It is quite apparent that many developers do not simply develop single pages — they build applications. Therefore, they need mechanics that deal with functionality throughout the entire application, not just the pages. One of the application capabilities provided by ASP.NET 3.5 is the site navigation system covered in this chapter. The underlying navigation system enables you to define your application’s navigation structure through an XML file, and it introduces a whole series of navigation server controls that work with the data from these XML files.
- **Chapter 15, “Personalization.”:** Developers are always looking for ways to store information pertinent to the end user. After it is stored, this personalization data has to be persisted for future visits or for grabbing other pages within the same application. The ASP.NET team developed a way to store this information — the ASP.NET personalization system. The great thing about this system is that you configure the entire behavior of the system from the `web.config` file.
- **Chapter 16, “Membership and Role Management.”:** This chapter covers the membership and role management system developed to simplify adding authentication and authorization to your ASP.NET applications. These two systems are extensive; they make some of the more complicated authentication and authorization implementations of the past a distant memory. This chapter focuses on using the `web.config` file for controlling how these systems are applied, as well as on the server controls that work with the underlying systems.
- **Chapter 17, “Portal Frameworks and Web Parts.”:** This chapter explains Web Parts — a way of encapsulating pages into smaller and more manageable objects. The great thing about Web Parts

is that they can be made of a larger Portal Framework, which can then enable end users to completely modify how the Web Parts are constructed on the page — including their appearance and layout.

- **Chapter 18, ''HTML and CSS Design with ASP.NET.'':** A lot of focus on building a CSS-based Web application was placed on Visual Studio 2008. This chapter takes a close look at how you can effectively work with HTML and CSS design for your ASP.NET applications.
- **Chapter 19, ''ASP.NET AJAX.'':** AJAX is a hot buzzword in the Web application world these days. AJAX is an acronym for *Asynchronous JavaScript and XML* and, in Web application development; it signifies the capability to build applications that make use of the `XMLHttpRequest` object. New to Visual Studio 2008 is the ability to build AJAX-enabled ASP.NET applications from the default install of the IDE. This chapter takes a look at a new way to build your applications.
- **Chapter 20, ''ASP.NET AJAX Control Toolkit.'':** Along with the new capabilities to build ASP.NET application which make use of the AJAX technology, there are a series of new controls that are now available to make the task rather simple. This chapter takes a good look at the ASP.NET AJAX Control Toolkit and how to use this toolkit with your applications today.
- **Chapter 21, ''Security.'':** This security chapter discusses security beyond the membership and role management features provided by ASP.NET 3.5. This chapter provides an in-depth look at the authentication and authorization mechanics inherent in the ASP.NET technology, as well as HTTP access types and impersonations.
- **Chapter 22, ''State Management.'':** Because ASP.NET is a request-response–based technology, state management and the performance of requests and responses take on significant importance. This chapter introduces these two separate but important areas of ASP.NET development.
- **Chapter 23, ''Caching.'':** Because of the request-response nature of ASP.NET, caching (storing previous generated results, images, and pages) on the server becomes rather important to the performance of your ASP.NET applications. This chapter looks at some of the advanced caching capabilities provided by ASP.NET, including the SQL cache invalidation feature which is part of ASP.NET 3.5.
- **Chapter 24, ''Debugging and Error Handling Techniques.'':** Being able to handle unanticipated errors in your ASP.NET applications is vital for any application that you build. This chapter tells you how to properly structure error handling within your applications. It also shows you how to use various debugging techniques to find errors that your applications might contain.
- **Chapter 25, ''File I/O and Streams.'':** More often than not, you want your ASP.NET applications to work with items that are outside the base application. Examples include files and streams. This chapter takes a close look at working with various file types and streams that might come into your ASP.NET applications.
- **Chapter 26, ''User and Server Controls.'':** Not only can you use the plethora of server controls that come with ASP.NET, but you can also utilize the same framework these controls use and build your own. This chapter describes building your own server controls and how to use them within your applications.
- **Chapter 27, ''Modules and Handlers.'':** Sometimes, just creating dynamic Web pages with the latest languages and databases does not give you, the developer, enough control over an application. At times, you need to be able to dig deeper and create applications that can interact with the Web server itself. You want to be able to interact with the low-level processes, such as how the Web server processes incoming and outgoing HTTP requests. This chapter looks at two methods of manipulating the way ASP.NET processes HTTP requests: HttpModule and HttpHandler.

Each method provides a unique level of access to the underlying processing of ASP.NET and can be powerful tools for creating web applications.

- **Chapter 28, ''Using Business Objects.'':** Invariably, you are going to have components created with previous technologies that you do not want to rebuild but that you do want to integrate into new ASP.NET applications. If this is the case, the .NET Framework makes it fairly simple and straightforward to incorporate your previous COM components into your applications. Beyond showing you how to integrate your COM components into your applications, this chapter also shows you how to build newer style .NET components instead of turning to the previous COM component architecture.
- **Chapter 29, ''Building and Consuming Services.'':** XML Web services have monopolized all the hype for the past few years, and a major aspect of the Web services model within .NET is part of ASP.NET. This chapter reveals the ease not only of building XML Web services, but consuming them in an ASP.NET application. This chapter then ventures further by describing how to build XML Web services that utilize SOAP headers and how to consume this particular type of service.
- **Chapter 30, ''Localization.'':** Developers usually build Web applications in the English language and then, as the audience for the application expands, they then realize the need to globalize the application. Of course, the ideal is to build the Web application to handle an international audience right from the start, but, in many cases, this may not be possible because of the extra work it requires. ASP.NET provides an outstanding way to address the internationalization of Web applications. You quickly realize that changes to the API, the addition of capabilities to the server controls, and even Visual Studio itself equip you to do the extra work required more easily to bring your application to an international audience. This chapter looks at some of the important items to consider when building your Web applications for the world.
- **Chapter 31, ''Configuration.'':** Configuration in ASP.NET can be a big topic because the ASP.NET team is not into building black boxes; instead, it is building the underlying capabilities of ASP.NET in a fashion that can easily be expanded on later. This chapter teaches you to modify the capabilities and behaviors of ASP.NET using the various configuration files at your disposal.
- **Chapter 32, ''Instrumentation.'':** ASP.NET 3.5 gives you greater capability to apply instrumentation techniques to your applications. The ASP.NET framework includes performance counters, the capability to work with the Windows Event Tracing system, possibilities for application tracing (covered in Chapter 24 of this book), and the most exciting part of this discussion — a health monitoring system that allows you to log a number of different events over an application's lifetime. This chapter takes an in-depth look at this health monitoring system.
- **Chapter 33, ''Administration and Management.'':** Besides making it easier for the developer to be more productive in building ASP.NET applications, the ASP.NET team also put considerable effort into making it easier to manage applications. In the past, using ASP.NET 1.0/1.1, you managed ASP.NET applications by changing values in an XML configuration file. This chapter provides an overview of the GUI tools that come with this release that enable you to manage your Web applications easily and effectively.
- **Chapter 34, ''Packaging and Deploying ASP.NET Applications.'':** So you have built an ASP.NET application-now what? This chapter takes the building process one-step further and shows you how to package your ASP.NET applications for easy deployment. Many options are available for working with the installers and compilation model to change what you are actually giving your customers.
- **Appendix A, ''Migrating Older ASP.NET Projects.'':** In some cases, you build your ASP.NET 3.5 applications from scratch, starting everything new. In many instances, however, this is not

an option. You need to take an ASP.NET application that was previously built on the 1.0, 1.1, or 2.0 versions of the .NET Framework and migrate the application so that it can run on the .NET Framework 3.5. This appendix focuses on migrating ASP.NET 1.x, or 2.0 applications to the 3.5 framework.

- ❑ **Appendix B, ''ASP.NET Ultimate Tools.'':** This chapter takes a look at the tools avilalable to you as an ASP.NET developer. Many of the tools here will help you to expedite your development process and in many cases, make you a better developer.
- ❑ **Appendix C, ''Silverlight.'':** Called WPF/E during its development days and now called Silverlight, this is a means to build fluid applications using XAML. This new technology enables developers with really rich vector-based applications.
- ❑ **Appendix D, ''ASP.NET Resources.'':** This small appendix points you to some of the more valuable online resources for enhancing your understanding of ASP.NET.

Conventions

This book uses a number of different styles of text and layout to help differentiate among various types of information. Here are examples of the styles used and an explanation of what they mean:

- ❑ New words being defined are shown in *italics*.
- ❑ Keys that you press on the keyboard, such as Ctrl and Enter, are shown in initial caps and spelled as they appear on the keyboard.
- ❑ File and folder names, file extensions, URLs, and code that appears in regular paragraph text are shown in a `monospaced` typeface.

When we show a block of code that you can type as a program and run, it's shown on separate lines, like this:

```
public static void Main()
{
   AFunc(1,2,"abc");
}
```

or like this:

```
public static void Main()
{
   AFunc(1,2,"abc");
}
```

Sometimes you see code in a mixture of styles, like this:

```
// If we haven't reached the end, return true, otherwise
// set the position to invalid, and return false.
pos++;

if (pos < 4)
   return true;

else {
```

```
            pos = -1;
            return false;
        }
```

When mixed code is shown like this, the code with the gray background is what you should focus on in the current example.

We demonstrate the syntactical usage of methods, properties, and so on using the following format:

```
SqlDependency="database:table"
```

Here, the italicized parts indicate *placeholder text:* object references, variables, or parameter values that you need to insert.

Most of the code examples throughout the book are presented as numbered listings that have descriptive titles, like this:

Listing I-7: Targeting WML devices in your ASP.NET pages

Each listing is numbered (for example: *1-3*) where the first number represents the chapter number and the number following the hyphen represents a sequential number that indicates where that listing falls within the chapter. Downloadable code from the Wrox Web site (`www.wrox.com`) also uses this numbering system so that you can easily locate the examples you are looking for.

All code is shown in both VB and C#, when warranted. The exception is for code in which the only difference is, for example, the value given to the `Language` attribute in the `Page` directive. In such situations, we don't repeat the code for the C# version; the code is shown only once, as in the following example:

```
<%@ Page Language="VB"%>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>DataSetDataSource</title>
</head>
<body>
    <form id="form1" runat="server">
       <asp:DropDownList ID="Dropdownlist1" Runat="server" DataTextField="name"
        DataSourceID="XmlDataSource1">
       </asp:DropDownList>

       <asp:XmlDataSource ID="XmlDataSource1" Runat="server"
        DataFile="~/Painters.xml">
       </asp:DataSetDataSource>
   </form>
</body>
</html>
```

Source Code

As you work through the examples in this book, you may choose either to type all the code manually or to use the source code files that accompany the book. All the source code used in this book is available for download at `www.wrox.com`. When you get to the site, simply locate the book's title (either by using the Search box or one of the topic lists) and click the Download Code link. You can then choose to download all the code from the book in one large zip file or download just the code you need for a particular chapter.

> *Because many books have similar titles, you may find it easiest to search by ISBN; this book's ISBN is 978-0-470-18757-9.*

After you download the code, just decompress it with your favorite compression tool. Alternatively, you can go to the main Wrox code download page at `www.wrox.com/dynamic/books/download.aspx` to see the code available for this book and all other Wrox books. Remember, you can easily find the code you are looking for by referencing the listing number of the code example from the book, such as ''Listing I-7.'' We used these listing numbers when naming the downloadable code files.

Errata

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, such as a spelling mistake or faulty piece of code, we would be very grateful if you'd tell us about it. By sending in errata, you may spare another reader hours of frustration; at the same time, you are helping us provide even higher-quality information.

To find the errata page for this book, go to `www.wrox.com` and locate the title using the Search box or one of the title lists. Then, on the book details page, click the Book Errata link. On this page, you can view all errata that have been submitted for this book and posted by Wrox editors. A complete book list including links to each book's errata is also available at `www.wrox.com/misc-pages/booklist.shtml`.

If you don't spot ''your'' error already on the Book Errata page, go to `www.wrox.com/contact/techsupport.shtml` and complete the form there to send us the error you have found. We'll check the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

p2p.wrox.com

For author and peer discussion, join the P2P forums at `p2p.wrox.com`. The forums are a Web-based system for you to post messages relating to Wrox books and technologies and to interact with other readers and technology users. The forums offer a subscription feature that enables you to receive e-mail on topics of interest when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are represented in these forums.

At `http://p2p.wrox.com` you will find a number of different forums that will help you not only as you read this book but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to `p2p.wrox.com` and click the Register link.
2. Read the terms of use and click Agree.
3. Supply the information required to join, as well as any optional information you want to provide, and click Submit.

You will receive an e-mail with information describing how to verify your account and complete the joining process.

You can read messages in the forums without joining P2P, but you must join in order to post messages.

After you join, you can post new messages and respond to other users' posts. You can read messages at any time on the Web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how the forum software works, as well as answers to many common questions specific to P2P and Wrox books, be sure to read the P2P FAQs. Simply click the FAQ link on any P2P page.

1

Application and Page Frameworks

The evolution of ASP.NET continues! The progression from Active Server Pages 3.0 to ASP.NET 1.0 was revolutionary, to say the least. And now the revolution continues with the latest release of ASP.NET — version 3.5. The original introduction of ASP.NET 1.0 fundamentally changed the Web programming model. ASP.NET 3.5 is just as revolutionary in the way it will increase your productivity. As of late, the primary goal of ASP.NET is to enable you to build powerful, secure, dynamic applications using the least possible amount of code. Although this book covers the new features provided by ASP.NET 3.5, it also covers all the offerings of ASP.NET technology.

If you are new to ASP.NET and building your first set of applications in ASP.NET 3.5, you may be amazed by the vast amount of wonderful server controls it provides. You may marvel at how it enables you to work with data more effectively using a series of data providers. You may be impressed at how easily you can build in security and personalization.

The outstanding capabilities of ASP.NET 3.5 do not end there, however. This chapter looks at many exciting options that facilitate working with ASP.NET pages and applications. One of the first steps you, the developer, should take when starting a project is to become familiar with the foundation you are building on and the options available for customizing that foundation.

Application Location Options

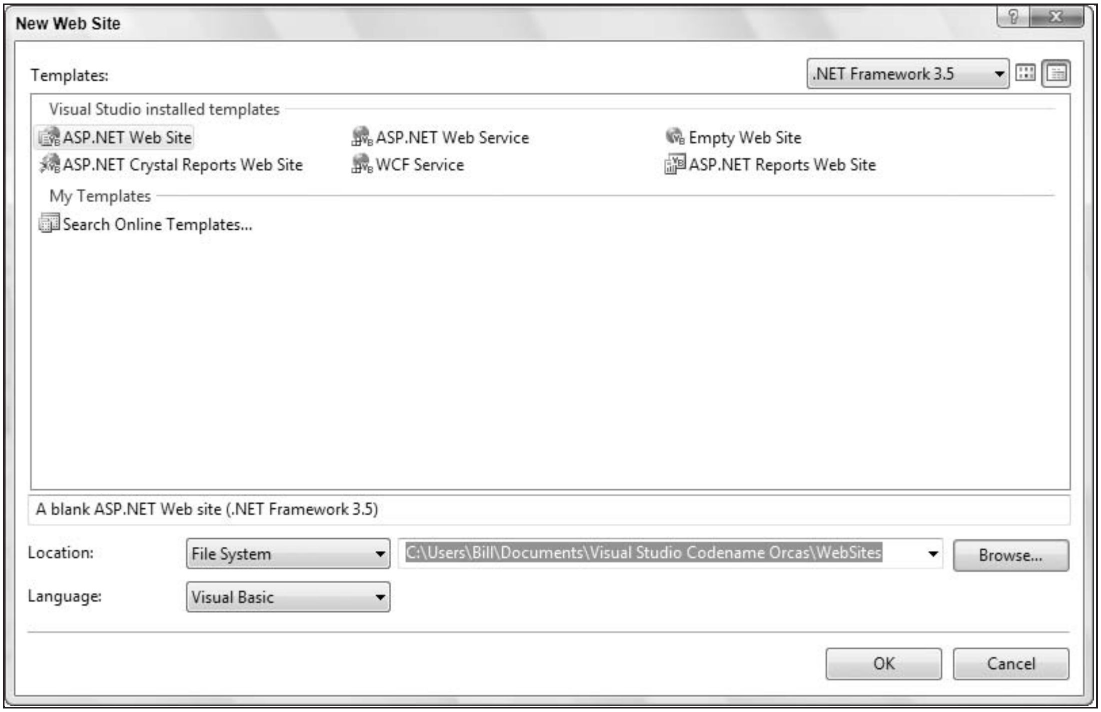
With ASP.NET 3.5, you have the option — using Visual Studio 2008 — to create an application with a virtual directory mapped to IIS or a standalone application outside the confines of IIS. Whereas the early Visual Studio .NET 2002/2003 IDEs forced developers to use IIS for all Web applications, Visual Studio 2008 (and Visual Web Developer 2008 Express Edition, for that matter) includes a built-in Web server that you can use for development, much like the one used in the past with the ASP.NET Web Matrix.

This built-in Web server was previously presented to developers as a code sample called Cassini. In fact, the code for this mini Web server is freely downloadable from the ASP.NET team Web site found at `www.asp.net`*.*

The following section shows you how to use the built-in Web server that comes with Visual Studio 2008.

Built-In Web Server

By default, Visual Studio 2008 builds applications without the use of IIS. You can see this when you select New ➪ Web Site in the IDE. By default, the location provided for your application is in `C:\Users\Bill\Documents\Visual Studio 2008\WebSites` if you are using Windows Vista (shown in Figure 1-1). It is not `C:\Inetpub\wwwroot\` as it would have been in Visual Studio .NET 2002/2003. By default, any site that you build and host inside `C:\Users\Bill\Documents\Visual Studio 2008\WebSites` (or any other folder you create) uses the built-in Web server that is part of Visual Studio 2008. If you use the built-in Web server from Visual Studio 2008, you are not locked into the `WebSites` folder; you can create any folder you want in your system.


New Web Site
Templates:
Visual Studio installed templates
ASP.NET Web Site
ASP.NET Web Service
Empty Web Site
ASP.NET Crystal Reports Web Site
WCF Service
ASP.NET Reports Web Site
My Templates
Search Online Templates...
Browse...
OK
Cancel

.NET Framework 3.5

A blank ASP.NET Web site (.NET Framework 3.5)

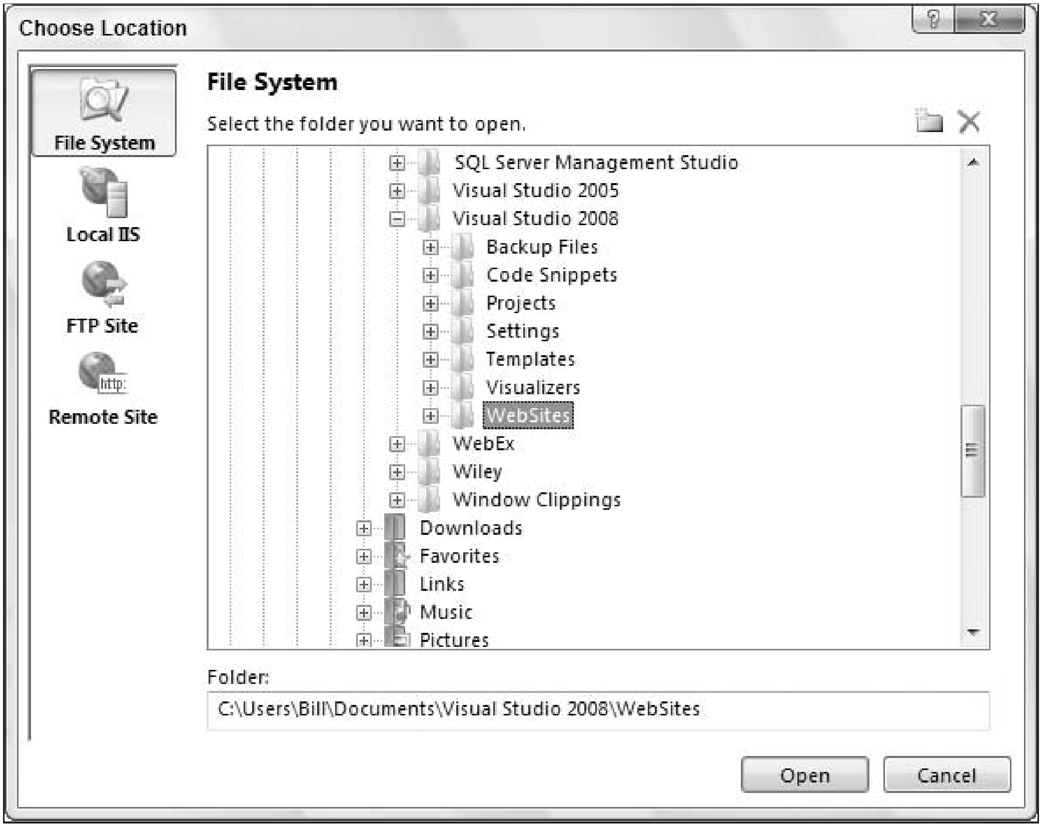
Location: File System C:\Users\Bill\Documents\Visual Studio Codename Orcas\WebSites

Language: Visual Basic

Figure 1-1

To change from this default, you have a handful of options. Click the Browse button in the New Web Site dialog. This brings up the Choose Location dialog, shown in Figure 1-2.

If you continue to use the built-in Web server that Visual Studio 2008 provides, you can choose a new location for your Web application from this dialog. To choose a new location, select a new folder and save your `.aspx` pages and any other associated files to this directory. When using Visual Studio 2008, you can run your application completely from this location. This way of working with the ASP.NET pages you create is ideal if you do not have access to a Web server because it enables you to build applications that do not reside on a machine with IIS. This means that you can even develop ASP.NET applications on operating systems such as Windows XP Home Edition.


Choose Location
File System
Local IIS
FTP Site
Remote Site
File System
Select the folder you want to open.
SQL Server Management Studio
Visual Studio 2005
Visual Studio 2008
Backup Files
Code Snippets
Projects
Settings
Templates
Visualizers
WebSites
WebEx
Wiley
Window Clippings
Downloads
Favorites
Links
Music
Pictures
Folder:
Open
Cancel

Folder: C:\Users\Bill\Documents\Visual Studio 2008\WebSites

Figure 1-2

IIS

From the Choose Location dialog, you can also change where your application is saved and which type of Web server your application employs. To use IIS (as you probably did when you used Visual Studio .NET 2002/2003), select the Local IIS button in the dialog. This changes the results in the text area to show you a list of all the virtual application roots on your machine. You are required to run Visual Studio as an administrator user if you want to see your local IIS instance.

To create a new virtual root for your application, highlight Default Web Site. Two accessible buttons appear at the top of the dialog box (see Figure 1-3). When you look from left to right, the first button in the upper-right corner of the dialog box is for creating a new Web application — or a virtual root. This button is shown as a globe inside a box. The second button enables you to create virtual directories for

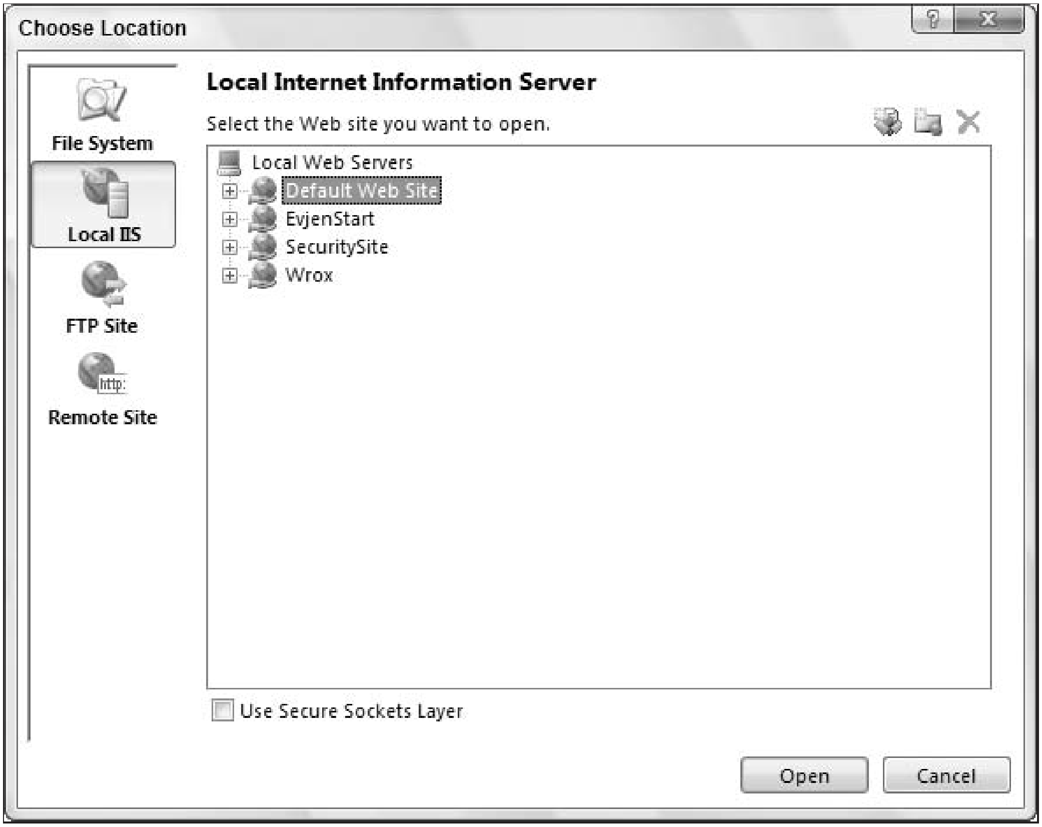

Choose Location
File System
Local IIS
FTP Site
Remote Site
Local Internet Information Server
Select the Web site you want to open.
Local Web Servers
Default Web Site
EvjenStart
SecuritySite
Wrox
☐ Use Secure Sockets Layer
Open
Cancel

Figure 1-3

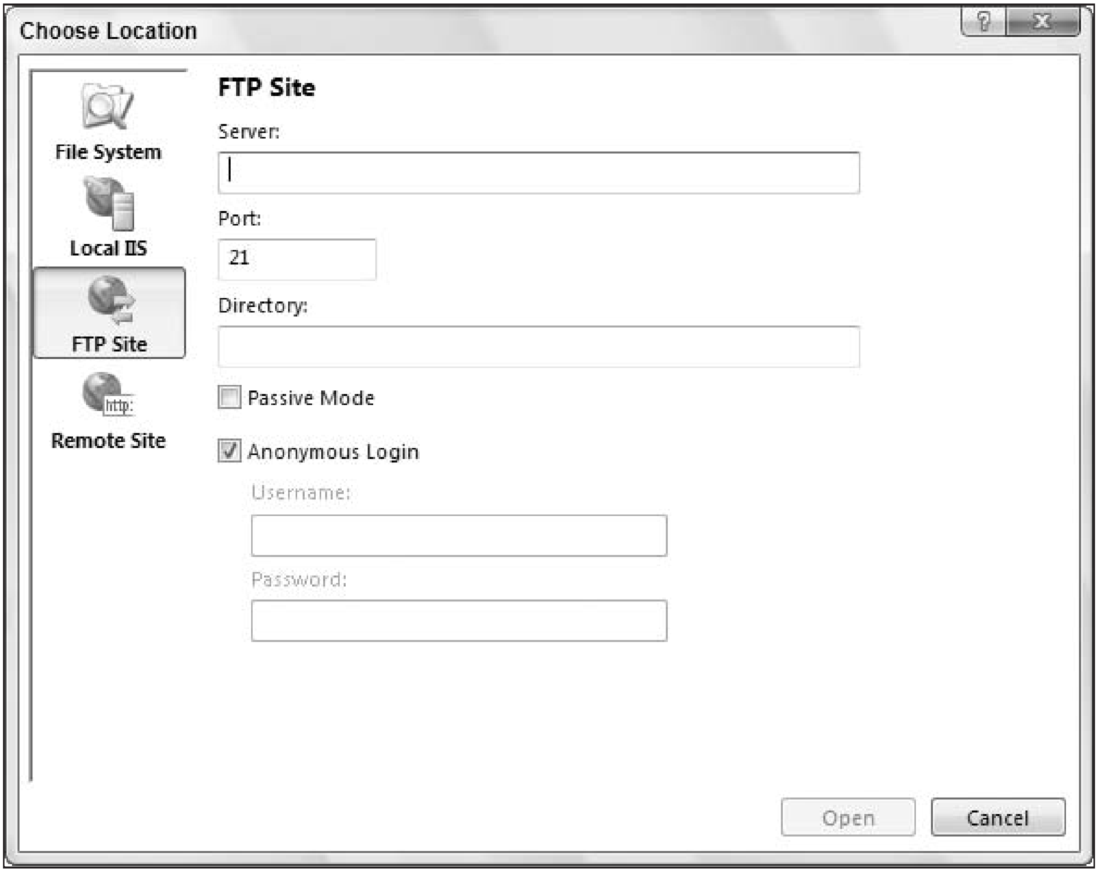
any of the virtual roots you created. The third button is a Delete button, which allows you to delete any selected virtual directories or virtual roots on the server.

After you have created the virtual directory you want, click the Open button. Visual Studio 2008 then goes through the standard process to create your application. Now, however, instead of depending on the built-in Web server from ASP.NET 3.5, your application will use IIS. When you invoke your application, the URL now consists of something like `http://localhost/MyWeb/Default.aspx`, which means it is using IIS.

FTP

Not only can you decide on the type of Web server for your Web application when you create it using the Choose Location dialog, but you can also decide where your application is going to be located. With the previous options, you built applications that resided on your local server. The FTP option enables you to actually store and even code your applications while they reside on a server somewhere else in your enterprise — or on the other side of the planet. You can also use the FTP capabilities to work on different locations within the same server. Using this new capability provides a wide range of possible options.

The built-in capability giving FTP access to your applications is a major enhancement to the IDE. Although formerly difficult to accomplish, this task is now quite simple, as you can see from Figure 1-4.



Server:

Port: 21

Directory:

☐ Passive Mode

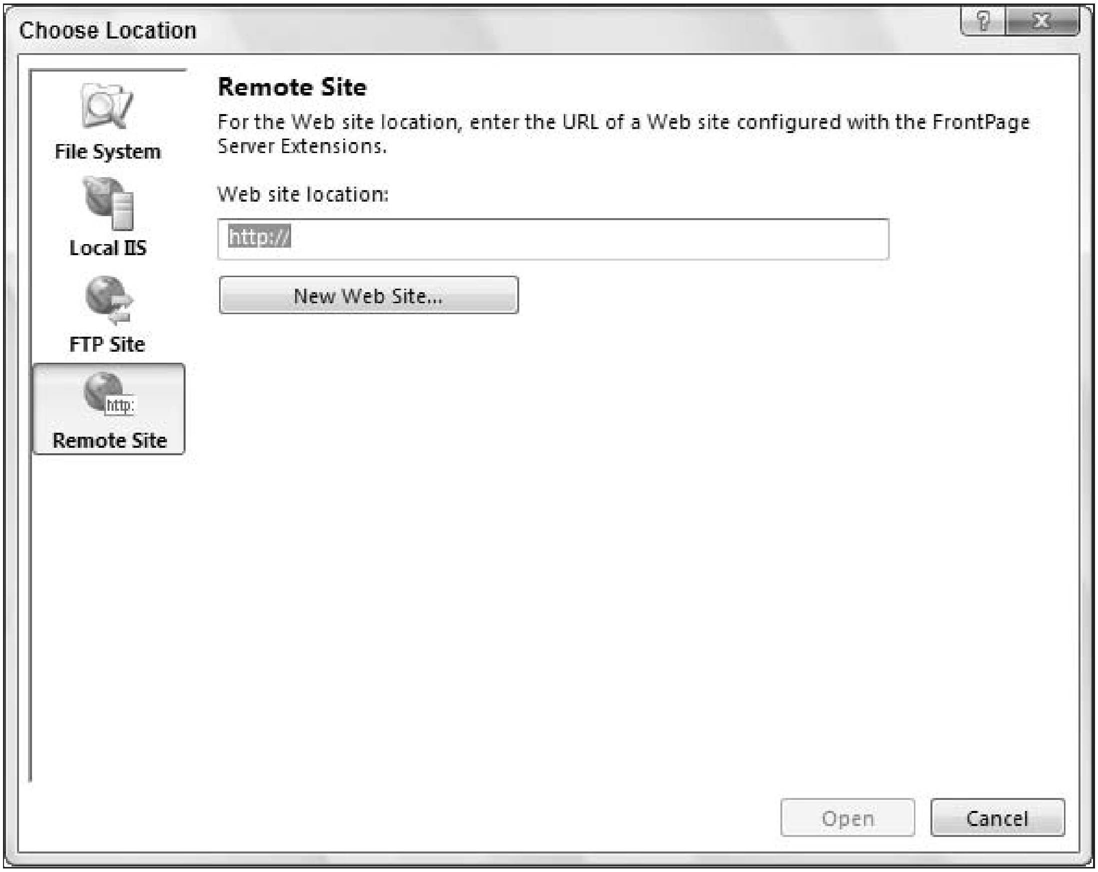
☑ Anonymous Login

Figure 1-4

To create your application on a remote server using FTP, simply provide the server name, the port to use, and the directory — as well as any required credentials. If the correct information is provided, Visual Studio 2008 reaches out to the remote server and creates the appropriate files for the start of your application, just as if it were doing the job locally. From this point on, you can open your project and connect to the remote server using FTP.

Web Site Requiring FrontPage Extensions

The last option in the Choose Location dialog is the Remote Sites option. Clicking this button provides a dialog that enables you to connect to a remote or local server that utilizes FrontPage Extensions. This option is displayed in Figure 1-5.



For the Web site location, enter the URL of a Web site configured with the FrontPage Server Extensions.

Web site location: http://

Figure 1-5

The ASP.NET Page Structure Options

ASP.NET 3.5 provides two paths for structuring the code of your ASP.NET pages. The first path utilizes the code-inline model. This model should be familiar to classic ASP 2.0/3.0 developers because all the code is contained within a single `.aspx` page. The second path uses ASP.NET's code-behind model, which allows for code separation of the page's business logic from its presentation logic. In this model, the presentation logic for the page is stored in an `.aspx` page, whereas the logic piece is stored in a separate class file: `.aspx.vb` or `.aspx.cs`. It is considered best practice to use the code-behind model as it provides a clean model in separation of pure UI elements from code that manipulates these elements. It is also seen as a better means in maintaining code.

One of the major complaints about Visual Studio .NET 2002 and 2003 is that it forced you to use the code-behind model when developing your ASP.NET pages because it did not understand the code-inline model. The code-behind model in ASP.NET was introduced as a new way to separate the presentation code and business logic. Listing 1-1 shows a typical `.aspx` page generated using Visual Studio .NET 2002 or 2003.

Listing 1-1: A typical .aspx page from ASP.NET 1.0/1.1

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.aspx.vb"
    Inherits="WebApplication.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
    <title>WebForm1</title>
    <meta name="GENERATOR" content="Microsoft Visual Studio .NET 7.1">
    <meta name="CODE_LANGUAGE" content="Visual Basic .NET 7.1">
    <meta name="vs_defaultClientScript" content="JavaScript">
    <meta name="vs_targetSchema"
      content="http://schemas.microsoft.com/intellisense/ie5">
 </HEAD>
 <body>
    <form id="Form1" method="post" runat="server">
       <P>What is your name?<br>
       <asp:TextBox id="TextBox1" runat="server"></asp:TextBox><BR>
       <asp:Button id="Button1" runat="server" Text="Submit"></asp:Button></P>
       <P><asp:Label id="Label1" runat="server"></asp:Label></P>
    </form>
 </body>
</HTML>
```

The code-behind file created within Visual Studio .NET 2002/2003 for the `.aspx` page is shown in Listing 1-2.

Listing 1-2: A typical .aspx.vb/.aspx.cs page from ASP.NET 1.0/1.1

```
Public Class WebForm1
    Inherits System.Web.UI.Page

#Region " Web Form Designer Generated Code "

    'This call is required by the Web Form Designer.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()

    End Sub
    Protected WithEvents TextBox1 As System.Web.UI.WebControls.TextBox
    Protected WithEvents Button1 As System.Web.UI.WebControls.Button
    Protected WithEvents Label1 As System.Web.UI.WebControls.Label

    'NOTE: The following placeholder declaration is required by the Web Form
        Designer.
    'Do not delete or move it.
    Private designerPlaceholderDeclaration As System.Object

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
      System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub
```

Continued

```
#End Region

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
      System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
      System.EventArgs) Handles Button1.Click
        Label1.Text = "Hello " & TextBox1.Text
    End Sub
End Class
```

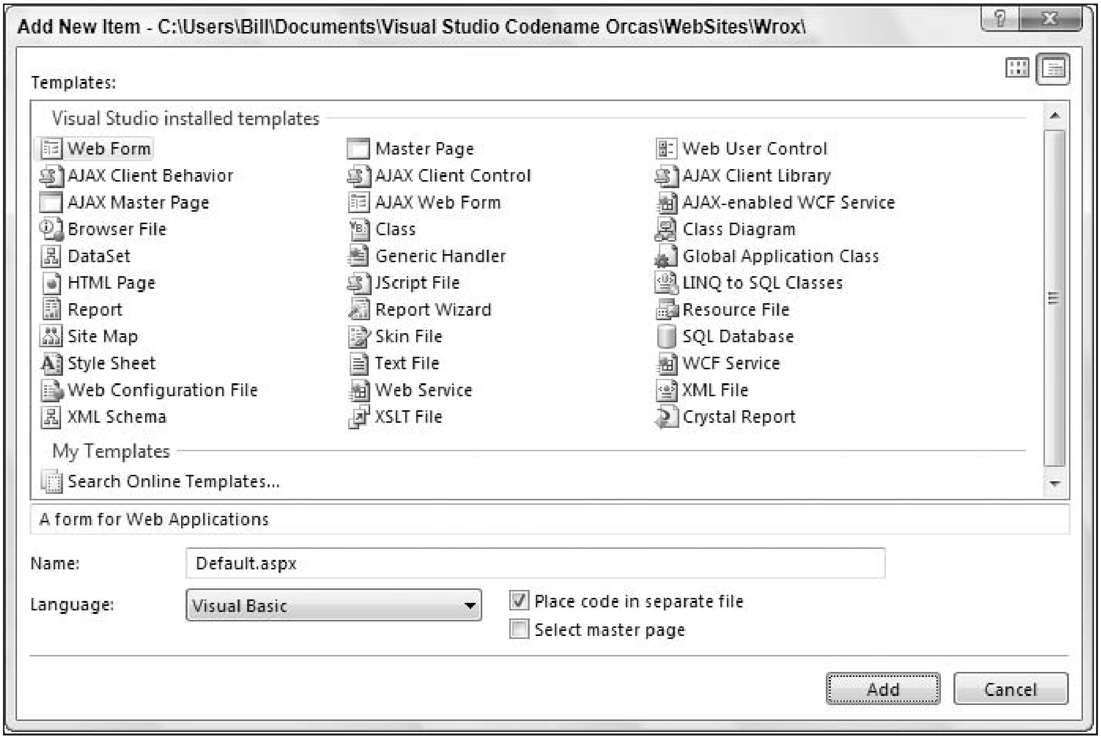
In this code-behind page from ASP.NET 1.0/1.1, you can see that a lot of the code that developers never have to deal with is hidden in the `#Region` section of the page. Because ASP.NET 3.5 is built on top of .NET 3.5, which in turn is utilizing the core .NET 2.0 Framework, it can take advantage of the .NET Framework capability of partial classes. Partial classes enable you to separate your classes into multiple class files, which are then combined into a single class when the application is compiled. Because ASP.NET 3.5 combines all this page code for you behind the scenes when the application is compiled, the code-behind files you work with in ASP.NET 3.5 are simpler in appearance and the model is easier to use. You are presented with only the pieces of the class that you need. Next, we will look at both the inline and code-behind models from ASP.NET 3.5.

Inline Coding

With the .NET Framework 1.0/1.1, developers went out of their way (and outside Visual Studio .NET) to build their ASP.NET pages inline and avoid the code-behind model that was so heavily promoted by Microsoft and others. Visual Studio 2008 (as well as Visual Web Developer 2008 Express Edition) allows you to build your pages easily using this coding style. To build an ASP.NET page inline instead of using the code-behind model, you simply select the page type from the Add New Item dialog and make sure that the Place Code in Separate File check box is unchecked. You can get at this dialog by right clicking the project or the solution in the Solution Explorer and selecting Add New Item (see Figure 1-6).

From here, you can see the check box you need to unselect if you want to build your ASP.NET pages inline. In fact, many page types have options for both inline and code-behind styles. The following table shows your inline options when selecting files from this dialog.

File Options Using Inline Coding	File Created
Web Form	`.aspx` file
AJAX Web Form	`.aspx` file
Master Page	`.master` file
AJAX Master Page	`.master` file
Web User Control	`.ascx` file
Web Service	`.asmx` file



Add New Item - C:\Users\Bill\Documents\Visual Studio Codename Orcas\WebSites\Wrox\

A form for Web Applications

Name: Default.aspx

Language: Visual Basic

☑ Place code in separate file

☐ Select master page

Templates:
Visual Studio installed templates
Web Form
AJAX Client Behavior
AJAX Master Page
Browser File
DataSet
HTML Page
Report
Site Map
Style Sheet
Web Configuration File
XML Schema
Master Page
AJAX Client Control
AJAX Web Form
Class
Generic Handler
JScript File
Report Wizard
Skin File
Text File
Web Service
XSLT File
Web User Control
AJAX Client Library
AJAX-enabled WCF Service
Class Diagram
Global Application Class
LINQ to SQL Classes
Resource File
SQL Database
WCF Service
XML File
Crystal Report
My Templates
Search Online Templates...
Add
Cancel

Figure 1-6

By using the Web Form option with a few controls, you get a page that encapsulates not only the presentation logic, but the business logic as well. This is illustrated in Listing 1-3.

Listing 1-3: A simple page that uses the inline coding model

VB

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Label1.Text = "Hello " & Textbox1.Text
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Simple Page</title>
</head>
```

Continued

```
<body>
    <form id="form1" runat="server">
        What is your name?<br />
        <asp:Textbox ID="Textbox1" Runat="server"></asp:Textbox><br />
        <asp:Button ID="Button1" Runat="server" Text="Submit"
         OnClick="Button1_Click" />
        <p><asp:Label ID="Label1" Runat="server"></asp:Label></p>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    protected void Button1_Click(object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + Textbox1.Text;
    }
</script>
```

From this example, you can see that all the business logic is encapsulated in between `<script>` tags. The nice feature of the inline model is that the business logic and the presentation logic are contained within the same file. Some developers find that having everything in a single viewable instance makes working with the ASP.NET page easier. Another great thing is that Visual Studio 2008 provides IntelliSense when working with the inline coding model and ASP.NET 3.5. Before Visual Studio 2005, this capability did not exist. Visual Studio .NET 2002/2003 forced you to use the code-behind model and, even if you rigged it so your pages were using the inline model, you lost all IntelliSense capabilities.

Code-Behind Model

The other option for constructing your ASP.NET 3.5 pages is to build your files using the code-behind model.

> *It is important to note that the more preferred method is the code-behind model rather than the inline model. This method employs the proper segmentation between presentation and business logic in many cases. You will find that many of the examples in this book use an inline coding model because it works well in showing an example in one listing. Even though the example is using an inline coding style, it is my recommendation that you move the code to employ the code-behind model.*

To create a new page in your ASP.NET solution that uses the code-behind model, select the page type you want from the New File dialog. To build a page that uses the code-behind model, you first select the page in the Add New Item dialog and make sure the Place Code in Separate File check box is checked. The following table shows you the options for pages that use the code-behind model.

File Options Using Code-Behind	File Created
Web Form	`.aspx` file `.aspx.vb` or `.aspx.cs` file
AJAX Web Form	`.aspx` file `.aspx.vb` or `.aspx.cs` file
Master Page	`.master` file `.master.vb` or `.master.cs` file
AJAX Master Page	`.master.vb` or `.master.cs` file
Web User Control	`.ascx` file `.ascx.vb` or `.ascx.cs` file
Web Service	`.asmx` file `.vb` or `.cs` file

The idea of using the code-behind model is to separate the business logic and presentation logic into separate files. Doing this makes it easier to work with your pages, especially if you are working in a team environment where visual designers work on the UI of the page and coders work on the business logic that sits behind the presentation pieces. In the earlier Listings 1-1 and 1-2, you saw how pages using the code-behind model in ASP.NET 1.0/1.1 were constructed. To see the difference in ASP.NET 3.5, look at how its code-behind pages are constructed. This is illustrated in Listing 1-4 for the presentation piece and Listing 1-5 for the code-behind piece.

Listing 1-4: An .aspx page that uses the ASP.NET 3.5 code-behind model

VB

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
    Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Simple Page</title>
</head>
<body>
    <form id="form1" runat="server">
        What is your name?<br />
        <asp:Textbox ID="Textbox1" Runat="server"></asp:Textbox><br />
        <asp:Button ID="Button1" Runat="server" Text="Submit"
         OnClick="Button1_Click" />
        <p><asp:Label ID="Label1" Runat="server"></asp:Label></p>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

Listing 1-5: A code-behind page

VB

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Button1_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs) Handles Button1.Click

        Label1.Text = "Hello " & TextBox1.Text
    End Sub
End Class
```

C#

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Xml.Linq;

public partial class _Default : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {
       Label1.Text = "Hello " + Textbox1.Text;
    }
}
```

The `.aspx` page using this ASP.NET 3.5 code-behind model has some attributes in the `Page` directive that you should pay attention to when working in this mode. The first is the `CodeFile` attribute. This is an attribute in the `Page` directive and is meant to point to the code-behind page that is used with this presentation page. In this case, the value assigned is `Default.aspx.vb` or `Default.aspx.cs`. The second attribute needed is the `Inherits` attribute. This attribute was available in previous versions of ASP.NET, but was little used before ASP.NET 2.0. This attribute specifies the name of the class that is bound to the page when the page is compiled. The directives are simple enough in ASP.NET 3.5. Look at the code-behind page from Listing 1-5.

The code-behind page is rather simple in appearance because of the partial class capabilities that .NET 3.5 provides. You can see that the class created in the code-behind file uses partial classes, employing the `Partial` keyword in Visual Basic 2008 and the `partial` keyword from C# 2008. This enables you to simply place the methods that you need in your page class. In this case, you have a button-click event and nothing else.

Later in this chapter, we look at the compilation process for both of these models.

ASP.NET 3.5 Page Directives

ASP.NET directives are something that is a part of every ASP.NET page. You can control the behavior of your ASP.NET pages by using these directives. Here is an example of the `Page` directive:

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
    Inherits="_Default" %>
```

Eleven directives are at your disposal in your ASP.NET pages or user controls. You use these directives in your applications whether the page uses the code-behind model or the inline coding model.

Basically, these directives are commands that the compiler uses when the page is compiled. Directives are simple to incorporate into your pages. A directive is written in the following format:

```
<%@ [Directive] [Attribute=Value] %>
```

From this, you can see that a directive is opened with a `<%@` and closed with a `%>`. It is best to put these directives at the top of your pages or controls because this is traditionally where developers expect to see them (although the page still compiles if the directives are located at a different place). Of course, you can also add more than a single attribute to your directive statements, as shown in the following:

```
<%@ [Directive] [Attribute=Value] [Attribute=Value] %>
```

The following table describes the directives at your disposal in ASP.NET 3.5.

Directive	Description
`Assembly`	Links an assembly to the Page or user control for which it is associated.
`Control`	Page directive meant for use with user controls (`.ascx`).
`Implements`	Implements a specified .NET Framework interface.
`Import`	Imports specified namespaces into the Page or user control.
`Master`	Enables you to specify master page–specific attributes and values to use when the page parses or compiles. This directive can be used only with master pages (`.master`).
`MasterType`	Associates a class name to a Page in order to get at strongly typed references or members contained within the specified master page.
`OutputCache`	Controls the output caching policies of a Page or user control.
`Page`	Enables you to specify page specific attributes and values to use when the page parses or compiles. This directive can be used only with ASP.NET pages (`.aspx`).
`PreviousPageType`	Enables an ASP.NET page to work with a postback from another page in the application.

Directive	Description
`Reference`	Links a Page or user control to the current Page or user control.
`Register`	Associates aliases with namespaces and class names for notation in custom server control syntax.

The following sections provide a quick review of each of these directives.

@Page

The `@Page` directive enables you to specify attributes and values for an ASP.NET page (`.aspx`) to be used when the page is parsed or compiled. This is the most frequently used directive of the bunch. Because the ASP.NET page is such an important part of ASP.NET, you have quite a few attributes at your disposal. The following table summarizes the attributes available through the `@Page` directive.

Attribute	Description
`AspCompat`	Permits the page to be executed on a single-threaded apartment thread when given a value of `True`. The default setting for this attribute is `False`.
`Async`	Specifies whether the ASP.NET page is processed synchronously or asynchronously.
`AsyncTimeout`	Specifies the amount of time in seconds to wait for the asynchronous task to complete. The default setting is 45 seconds. This is a new attribute of ASP.NET 3.5.
`AutoEventWireup`	Specifies whether the page events are autowired when set to `True`. The default setting for this attribute is `True`.
`Buffer`	Enables HTTP response buffering when set to `True`. The default setting for this attribute is `True`.
`ClassName`	Specifies the name of the class that is bound to the page when the page is compiled.
`ClientTarget`	Specifies the target user agent a control should render content for. This attribute needs to be tied to an alias defined in the `<clientTarget>` section of the web.config.
`CodeFile`	References the code-behind file with which the page is associated.
`CodeFileBaseClass`	Specifies the type name of the base class to use with the code-behind class, which is used by the CodeFile attribute.
`CodePage`	Indicates the code page value for the response.
`CompilationMode`	Specifies whether ASP.NET should compile the page or not. The available options include `Always` (the default), `Auto`, or `Never`. A setting of Auto means that if possible, ASP.NET will not compile the page.

Attribute	Description
`CompilerOptions`	Compiler string that indicates compilation options for the page.
`CompileWith`	Takes a `String` value that points to the code-behind file used.
`ContentType`	Defines the HTTP content type of the response as a standard MIME type.
`Culture`	Specifies the culture setting of the page. ASP.NET 3.5 includes the capability to give the `Culture` attribute a value of `Auto` to enable automatic detection of the culture required.
`Debug`	Compiles the page with debug symbols in place when set to `True`.
`Description`	Provides a text description of the page. The ASP.NET parser ignores this attribute and its assigned value.
`EnableEventValidation`	Specifies whether to enable validation of events in postback and callback scenarios. The default setting of `True` means that events will be validated.
`EnableSessionState`	Session state for the page is enabled when set to `True`. The default setting is `True`.
`EnableTheming`	Page is enabled to use theming when set to `True`. The default setting for this attribute is `True`.
`EnableViewState`	View state is maintained across the page when set to `True`. The default value is `True`.
`EnableViewStateMac`	Page runs a machine-authentication check on the page's view state when the page is posted back from the user when set to `True`. The default value is `False`.
`ErrorPage`	Specifies a URL to post to for all unhandled page exceptions.
`Explicit`	Visual Basic `Explicit` option is enabled when set to `True`. The default setting is `False`.
`Language`	Defines the language being used for any inline rendering and script blocks.
`LCID`	Defines the locale identifier for the Web Form's page.
`LinePragmas`	`Boolean` value that specifies whether line pragmas are used with the resulting assembly.
`MasterPageFile`	Takes a `String` value that points to the location of the master page used with the page. This attribute is used with content pages.
`MaintainScrollPositionOn` `Postback`	Takes a `Boolean` value, which indicates whether the page should be positioned exactly in the same scroll position or if the page should be regenerated in the uppermost position for when the page is posted back to itself.

Attribute	Description
`ResponseEncoding`	Specifies the response encoding of the page content.
`SmartNavigation`	Specifies whether to activate the ASP.NET Smart Navigation feature for richer browsers. This returns the postback to the current position on the page. The default value is `False`.
`Src`	Points to the source file of the class used for the code behind of the page being rendered.
`Strict`	Compiles the page using the Visual Basic `Strict` mode when set to `True`. The default setting is `False`.
`StylesheetTheme`	Applies the specified theme to the page using the ASP.NET 3.5 themes feature. The difference between the `StylesheetTheme` and `Theme` attributes is that `StylesheetTheme` will not override preexisting style settings in the controls, whereas `Theme` will remove these settings.
`Theme`	Applies the specified theme to the page using the ASP.NET 3.5 themes feature.
`Title`	Applies a page's title. This is an attribute mainly meant for content pages that must apply a page title other than what is specified in the master page.
`Trace`	Page tracing is enabled when set to `True`. The default setting is `False`.
`TraceMode`	Specifies how the trace messages are displayed when tracing is enabled. The settings for this attribute include `SortByTime` or `SortByCategory`. The default setting is `SortByTime`.
`Transaction`	Specifies whether transactions are supported on the page. The settings for this attribute are `Disabled`, `NotSupported`, `Supported`, `Required`, and `RequiresNew`. The default setting is `Disabled`.
`UICulture`	The value of the `UICulture` attribute specifies what UI Culture to use for the ASP.NET page. ASP.NET 3.5 includes the capability to give the `UICulture` attribute a value of `Auto` to enable automatic detection of the `UICulture`.
`ValidateRequest`	When this attribute is set to `True`, the form input values are checked against a list of potentially dangerous values. This helps protect your Web application from harmful attacks such as JavaScript attacks. The default value is `True`.
`ViewStateEncryptionMode`	Specifies how the ViewState is encrypted on the page. The options include `Auto`, `Always`, and `Never`. The default is `Auto`.
`WarningLevel`	Specifies the compiler warning level at which to stop compilation of the page. Possible values are `0` through `4`.

Here is an example of how to use the `@Page` directive:

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
    Inherits="_Default" %>
```

@Master

The `@Master` directive is quite similar to the `@Page` directive except that the `@Master` directive is meant for master pages (`.master`). In using the `@Master` directive, you specify properties of the templated page that you will be using in conjunction with any number of content pages on your site. Any content pages (built using the `@Page` directive) can then inherit from the master page all the master content (defined in the master page using the `@Master` directive). Although they are similar, the `@Master` directive has fewer attributes available to it than does the `@Page` directive. The available attributes for the `@Master` directive are shown in the following table.

Attribute	**Description**
`AutoEventWireup`	Specifies whether the master page's events are autowired when set to `True`. Default setting is `True`.
`ClassName`	Specifies the name of the class that is bound to the master page when compiled.
`CodeFile`	References the code-behind file with which the page is associated.
`CompilationMode`	Specifies whether ASP.NET should compile the page or not. The available options include `Always` (the default), `Auto`, or `Never`. A setting of Auto means that if possible, ASP.NET will not compile the page.
`CompilerOptions`	Compiler string that indicates compilation options for the master page.
`CompileWith`	Takes a `String` value that points to the code-behind file used for the master page.
`Debug`	Compiles the master page with debug symbols in place when set to `True`.
`Description`	Provides a text description of the master page. The ASP.NET parser ignores this attribute and its assigned value.
`EnableTheming`	Indicates the master page is enabled to use theming when set to `True`. The default setting for this attribute is `True`.
`EnableViewState`	Maintains view state for the master page when set to `True`. The default value is `True`.
`Explicit`	Indicates that the Visual Basic `Explicit` option is enabled when set to `True`. The default setting is `False`.
`Inherits`	Specifies the `CodeBehind` class for the master page to inherit.
`Language`	Defines the language that is being used for any inline rendering and script blocks.

Attribute	Description
`LinePragmas`	`Boolean` value that specifies whether line pragmas are used with the resulting assembly.
`MasterPageFile`	Takes a `String` value that points to the location of the master page used with the master page. It is possible to have a master page use another master page, which creates a nested master page.
`Src`	Points to the source file of the class used for the code behind of the master page being rendered.
`Strict`	Compiles the master page using the Visual Basic `Strict` mode when set to `True`. The default setting is `False`.
`WarningLevel`	Specifies the compiler warning level at which you want to abort compilation of the page. Possible values are from `0` to `4`.

Here is an example of how to use the `@Master` directive:

```
<%@ Master Language="VB" CodeFile="MasterPage1.master.vb"
    AutoEventWireup="false" Inherits="MasterPage" %>
```

@Control

The `@Control` directive is similar to the `@Page` directive except that `@Control` is used when you build an ASP.NET user control. The `@Control` directive allows you to define the properties to be inherited by the user control. These values are assigned to the user control as the page is parsed and compiled. The available attributes are fewer than those of the `@Page` directive, but quite a few of them allow for the modifications you need when building user controls. The following table details the available attributes.

Attribute	Description
`AutoEventWireup`	Specifies whether the user control's events are autowired when set to `True`. Default setting is `True`.
`ClassName`	Specifies the name of the class that is bound to the user control when the page is compiled.
`CodeFileBaseClass`	Specifies the type name of the base class to use with the code-behind class, which is used by the CodeFile attribute.
`CodeFile`	References the code-behind file with which the user control is associated.
`CompilerOptions`	Compiler string that indicates compilation options for the user control.
`CompileWith`	Takes a `String` value that points to the code-behind file used for the user control.
`Debug`	Compiles the user control with debug symbols in place when set to `True`.

Attribute	Description
`Description`	Provides a text description of the user control. The ASP.NET parser ignores this attribute and its assigned value.
`EnableTheming`	User control is enabled to use theming when set to `True`. The default setting for this attribute is `True`.
`EnableViewState`	View state is maintained for the user control when set to `True`. The default value is `True`.
`Explicit`	Visual Basic `Explicit` option is enabled when set to `True`. The default setting is `False`.
`Inherits`	Specifies the `CodeBehind` class for the user control to inherit.
`Language`	Defines the language used for any inline rendering and script blocks.
`LinePragmas`	`Boolean` value that specifies whether line pragmas are used with the resulting assembly.
`Src`	Points to the source file of the class used for the code behind of the user control being rendered.
`Strict`	Compiles the user control using the Visual Basic `Strict` mode when set to `True`. The default setting is `False`.
`WarningLevel`	Specifies the compiler warning level at which to stop compilation of the user control. Possible values are `0` through `4`.

The `@Control` directive is meant to be used with an ASP.NET user control. The following is an example of how to use the directive:

```
<%@ Control Language="VB" Explicit="True"
    CodeFile="WebUserControl.ascx.vb" Inherits="WebUserControl"
    Description="This is the registration user control." %>
```

@Import

The `@Import` directive allows you to specify a namespace to be imported into the ASP.NET page or user control. By importing, all the classes and interfaces of the namespace are made available to the page or user control. This directive supports only a single attribute: `Namespace`.

The `Namespace` attribute takes a `String` value that specifies the namespace to be imported. The `@Import` directive cannot contain more than one attribute/value pair. Because of this, you must place multiple namespace imports in multiple lines as shown in the following example:

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
```

Several assemblies are already being referenced by your application. You can find a list of these imported namespaces by looking in the root `web.config` file found at `C:\Windows\Microsoft.NET\Framework\`

`v2.0.50727\CONFIG`. You can find this list of assemblies being referenced from the `<assemblies>` child element of the `<compilation>` element. The settings in the root `web.config` file are as follows:

```
<assemblies>
   <add assembly="mscorlib"/>
   <add assembly="System, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089"/>
   <add assembly="System.Configuration, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a"/>
   <add assembly="System.Web, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a"/>
   <add assembly="System.Data, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089"/>
   <add assembly="System.Web.Services, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a"/>
   <add assembly="System.Xml, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089"/>
   <add assembly="System.Drawing, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a"/>
   <add assembly="System.EnterpriseServices, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a"/>
   <add assembly="System.Web.Mobile, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a"/>
   <add assembly="*"/>
   <add assembly="System.Runtime.Serialization, Version=3.0.0.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089, processorArchitecture=MSIL"/>
   <add assembly="System.IdentityModel, Version=3.0.0.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089, processorArchitecture=MSIL"/>
   <add assembly="System.ServiceModel, Version=3.0.0.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089"/>
   <add assembly="System.ServiceModel.Web, Version=3.5.0.0, Culture=neutral,
      PublicKeyToken=31bf3856ad364e35"/>
   <add assembly="System.WorkflowServices, Version=3.5.0.0, Culture=neutral,
      PublicKeyToken=31bf3856ad364e35"/>
</assemblies>
```

Because of this reference in the root `web.config` file, these assemblies need not be referenced in a References folder, as you would have done in ASP.NET 1.0/1.1. You can actually add or delete assemblies that are referenced from this list. For example, if you have a custom assembly referenced continuously by each and every application on the server, you can simply add a similar reference to your custom assembly next to these others. Note that you can perform this same task through the application-specific `web.config` file of your application as well.

Even though assemblies might be referenced, you must still import the namespaces of these assemblies into your pages. The same root `web.config` file contains a list of namespaces automatically imported into each and every page of your application. This is specified through the `<namespaces>` child element of the `<pages>` element.

```
<namespaces>
   <add namespace="System" />
   <add namespace="System.Collections" />
   <add namespace="System.Collections.Specialized" />
   <add namespace="System.Configuration" />
   <add namespace="System.Text" />
```

```
      <add namespace="System.Text.RegularExpressions" />
      <add namespace="System.Web" />
      <add namespace="System.Web.Caching" />
      <add namespace="System.Web.SessionState" />
      <add namespace="System.Web.Security" />
      <add namespace="System.Web.Profile" />
      <add namespace="System.Web.UI" />
      <add namespace="System.Web.UI.WebControls" />
      <add namespace="System.Web.UI.WebControls.WebParts" />
      <add namespace="System.Web.UI.HtmlControls" />
   </namespaces>
```

From this XML list, you can see that quite a number of namespaces are imported into each and every one of your ASP.NET pages. Again, you can feel free to modify this selection in the root `web.config` file or even make a similar selection of namespaces from within your application's `web.config` file.

For instance, you can import your own namespace in the `web.config` file of your application in order to make the namespace available on every page where it is utilized.

```
<?xml version="1.0"?>
<configuration>
   <system.web>
      <pages>
         <namespaces>
            <add namespace="MyCompany.Utilities" />
         </namespaces>
      </pages>
   </system.web>
</configuration>
```

Remember that importing a namespace into your ASP.NET page or user control gives you the opportunity to use the classes without fully identifying the class name. For example, by importing the namespace `System.Data.OleDb` into the ASP.NET page, you can refer to classes within this namespace by using the singular class name (`OleDbConnection` instead of `System.Data.OleDb.OleDbConnection`).

@Implements

The `@Implements` directive gets the ASP.NET page to implement a specified .NET Framework interface. This directive supports only a single attribute: `Interface`.

The `Interface` attribute directly specifies the .NET Framework interface. When the ASP.NET page or user control implements an interface, it has direct access to all its events, methods, and properties.

Here is an example of the `@Implements` directive:

```
<%@ Implements Interface="System.Web.UI.IValidator" %>
```

@Register

The `@Register` directive associates aliases with namespaces and class names for notation in custom server control syntax. You can see the use of the `@Register` directive when you drag and drop a user control onto any of your `.aspx` pages. Dragging a user control onto the `.aspx` page causes Visual Studio 2008 to create an `@Register` directive at the top of the page. This registers your user control on the page so that the control can then be accessed on the `.aspx` page by a specific name.

The `@Register` directive supports five attributes, as described in the following table.

Attribute	Description
`Assembly`	The assembly you are associating with the `TagPrefix`.
`Namespace`	The namespace to relate with `TagPrefix`.
`Src`	The location of the user control.
`TagName`	The alias to relate to the class name.
`TagPrefix`	The alias to relate to the namespace.

Here is an example of how to use the `@Register` directive to import a user control to an ASP.NET page:

```
<%@ Register TagPrefix="MyTag" Namespace="MyName.MyNamespace"
   Assembly="MyAssembly" %>
```

@Assembly

The `@Assembly` directive attaches assemblies, the building blocks of .NET applications, to an ASP.NET page or user control as it compiles, thereby making all the assembly's classes and interfaces available to the page. This directive supports two attributes: `Name` and `Src`.

- ❑ `Name`: Enables you to specify the name of an assembly used to attach to the page files. The name of the assembly should include the file name only, not the file's extension. For instance, if the file is `MyAssembly.vb`, the value of the name attribute should be `MyAssembly`.
- ❑ `Src`: Enables you to specify the source of the assembly file to use in compilation.

The following provides some examples of how to use the `@Assembly` directive:

```
<%@ Assembly Name="MyAssembly" %>
<%@ Assembly Src="MyAssembly.vb" %>
```

@PreviousPageType

This directive is used to specify the page from which any cross-page postings originate. Cross-page posting between ASP.NET pages is explained later in the section ''Cross-Page Posting'' and again in Chapter 17.

The `@PreviousPageType` directive is a new directive that works with the new cross-page posting capability that ASP.NET 3.5 provides. This simple directive contains only two possible attributes: `TypeName` and `VirtualPath`:

- ❑ `TypeName`: Sets the name of the derived class from which the postback will occur.
- ❑ `VirtualPath`: Sets the location of the posting page from which the postback will occur.

@MasterType

The `@MasterType` directive associates a class name to an ASP.NET page in order to get at strongly typed references or members contained within the specified master page. This directive supports two attributes:

- ❑ `TypeName`: Sets the name of the derived class from which to get strongly typed references or members.
- ❑ `VirtualPath`: Sets the location of the page from which these strongly typed references and members will be retrieved.

Details of how to use the `@MasterType` directive are shown in Chapter 8. Here is an example of its use:

```
<%@ MasterType VirtualPath="~/Wrox.master" %>
```

@OutputCache

The `@OutputCache` directive controls the output caching policies of an ASP.NET page or user control. This directive supports the ten attributes described in the following table.

Attribute	**Description**
`CacheProfile`	Allows for a central way to manage an application's cache profile. Use the `CacheProfile` attribute to specify the name of the cache profile detailed in the `web.config`.
`Duration`	The duration of time in seconds that the ASP.NET page or user control is cached.
`Location`	Location enumeration value. The default is `Any`. This is valid for `.aspx` pages only and does not work with user controls (`.ascx`). Other possible values include `Client`, `Downstream`, `None`, `Server`, and `ServerAndClient`.
`NoStore`	Specifies whether to send a no-store header with the page.
`Shared`	Specifies whether a user control's output can be shared across multiple pages. This attribute takes a `Boolean` value and the default setting is `false`.
`SqlDependency`	Enables a particular page to use SQL Server cache invalidation.
`VaryByControl`	Semicolon-separated list of strings used to vary the output cache of a user control.
`VaryByCustom`	String specifying the custom output caching requirements.
`VaryByHeader`	Semicolon-separated list of HTTP headers used to vary the output cache.
`VaryByParam`	Semicolon-separated list of strings used to vary the output cache.

Here is an example of how to use the `@OutputCache` directive:

```
<%@ OutputCache Duration="180" VaryByParam="None" %>
```

Remember that the `Duration` attribute specifies the amount of time in *seconds* during which this page is to be stored in the system cache.

@Reference

The `@Reference` directive declares that another ASP.NET page or user control should be compiled along with the active page or control. This directive supports just a single attribute:

- `VirtualPath`: Sets the location of the page or user control from which the active page will be referenced.

Here is an example of how to use the `@Reference` directive:

```
<%@ Reference VirtualPath="~/MyControl.ascx" %>
```

ASP.NET Page Events

ASP.NET developers consistently work with various events in their server-side code. Many of the events that they work with pertain to specific server controls. For instance, if you want to initiate some action when the end user clicks a button on your Web page, you create a button-click event in your server-side code, as shown in Listing 1-6.

Listing 1-6: A sample button-click event shown in VB

```
Protected Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
   Label1.Text = TextBox1.Text
End Sub
```

In addition to the server controls, developers also want to initiate actions at specific moments when the ASP.NET page is being either created or destroyed. The ASP.NET page itself has always had a number of events for these instances. The following list shows you all the page events you could use in ASP.NET 1.0/1.1:

- `AbortTransaction`
- `CommitTransaction`
- `DataBinding`
- `Disposed`
- `Error`
- `Init`
- `Load`
- `PreRender`
- `Unload`

One of the more popular page events from this list is the `Load` event, which is used in VB as shown in Listing 1-7.

Listing 1-7: Using the Page_Load event

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   Handles Me.Load

   Response.Write("This is the Page_Load event")
End Sub
```

Besides the page events just shown, ASP.NET 3.5 has the following events:

- ❑ `InitComplete`: Indicates the initialization of the page is completed.
- ❑ `LoadComplete`: Indicates the page has been completely loaded into memory.
- ❑ `PreInit`: Indicates the moment immediately before a page is initialized.
- ❑ `PreLoad`: Indicates the moment before a page has been loaded into memory.
- ❑ `PreRenderComplete`: Indicates the moment directly before a page has been rendered in the browser.

An example of using any of these events, such as the `PreInit` event, is shown in Listing 1-8.

Listing 1-8: Using the new page events

VB

```
<script runat="server" language="vb">
    Protected Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs)
       Page.Theme = Request.QueryString("ThemeChange")
    End Sub
</script>
```

C#

```
<script runat="server">
    protected void Page_PreInit(object sender, System.EventArgs e)
    {
       Page.Theme = Request.QueryString["ThemeChange"];
    }
</script>
```

If you create an ASP.NET 3.5 page and turn on tracing, you can see the order in which the main page events are initiated. They are fired in the following order:

1. `PreInit`
2. `Init`
3. `InitComplete`
4. `PreLoad`
5. `Load`
6. `LoadComplete`

7. `PreRender`
8. `PreRenderComplete`
9. `Unload`

With the addition of these choices, you can now work with the page and the controls on the page at many different points in the page-compilation process. You see these useful new page events in code examples throughout the book.

Dealing with PostBacks

When you are working with ASP.NET pages, be sure you understand the page events just listed. They are important because you place a lot of your page behavior inside these events at specific points in a page lifecycle.

In Active Server Pages 3.0, developers had their pages post to other pages within the application. ASP.NET pages typically post back to themselves in order to process events (such as a button-click event).

For this reason, you must differentiate between posts for the first time a page is loaded by the end user and *postbacks*. A postback is just that — a posting back to the same page. The postback contains all the form information collected on the initial page for processing if required.

Because of all the postbacks that can occur with an ASP.NET page, you want to know whether a request is the first instance for a particular page or is a postback from the same page. You can make this check by using the `IsPostBack` property of the `Page` class, as shown in the following example:

VB

```
If Page.IsPostBack = True Then
   ' Do processing
End If
```

C#

```
if (Page.IsPostBack == true) {
   // Do processing
}
```

In addition to checking against a `True` or `False` value, you can also find out if the request is not a postback in the following manner:

VB

```
If Not Page.IsPostBack Then
   ' Do processing
End If
```

C#

```
if (!Page.IsPostBack) {
   // Do processing
}
```

Cross-Page Posting

One common feature in ASP 3.0 that is difficult to achieve in ASP.NET 1.0/1.1 is the capability to do cross-page posting. Cross-page posting enables you to submit a form (say, `Page1.aspx`) and have this form and all the control values post themselves to another page (`Page2.aspx`).

Traditionally, any page created in ASP.NET 1.0/1.1 simply posted to itself, and you handled the control values within this page instance. You could differentiate between the page's first request and any postbacks by using the `Page.IsPostBack` property, as shown here:

```
If Page.IsPostBack Then
   ' deal with control values
End If
```

Even with this capability, many developers still wanted to be able to post to another page and deal with the first page's control values on that page. This is something that is possible in ASP.NET 3.5, and it is quite a simple process.

For an example, create a page called `Page1.aspx` that contains a simple form. This page is shown in Listing 1-9.

Listing 1-9: Page1.aspx

VB

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Label1.Text = "Hello " & TextBox1.Text & "<br />" & _
            "Date Selected: " & Calendar1.SelectedDate.ToShortDateString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>First Page</title>
</head>
<body>
    <form id="form1" runat="server">
        Enter your name:<br />
        <asp:Textbox ID="TextBox1" Runat="server">
        </asp:Textbox>
        <p>
        When do you want to fly?<br />
        <asp:Calendar ID="Calendar1" Runat="server"></asp:Calendar></p>
        <br />
        <asp:Button ID="Button1" Runat="server" Text="Submit page to itself"
```

Continued

```
         OnClick="Button1_Click" />
        <asp:Button ID="Button2" Runat="server" Text="Submit page to Page2.aspx"
         PostBackUrl="~/Page2.aspx" />
        <p>
        <asp:Label ID="Label1" Runat="server"></asp:Label></p>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click (object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + TextBox1.Text + "<br />" +
            "Date Selected: " + Calendar1.SelectedDate.ToShortDateString();
    }
</script>
```

The code from `Page1.aspx`, as shown in Listing 1-9, is quite interesting. First, two buttons are shown on the page. Both buttons submit the form, but each submits the form to a different location. The first button submits the form to itself. This is the behavior that has been the default for ASP.NET 1.0/1.1. In fact, nothing is different about `Button1`. It submits to `Page1.aspx` as a postback because of the use of the `OnClick` property in the button control. A `Button1_Click` method on `Page1.aspx` handles the values that are contained within the server controls on the page.

The second button, `Button2`, works quite differently. This button does not contain an `OnClick` method as the first button did. Instead, it uses the `PostBackUrl` property. This property takes a string value that points to the location of the file to which this page should post. In this case, it is `Page2.aspx`. This means that `Page2.aspx` now receives the postback and all the values contained in the `Page1.aspx` controls. Look at the code for `Page2.aspx`, shown in Listing 1-10.

Listing 1-10: Page2.aspx

VB

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim pp_Textbox1 As TextBox
        Dim pp_Calendar1 As Calendar

        pp_Textbox1 = CType(PreviousPage.FindControl("Textbox1"), TextBox)
        pp_Calendar1 = CType(PreviousPage.FindControl("Calendar1"), Calendar)

        Label1.Text = "Hello " & pp_Textbox1.Text & "<br />" & _
            "Date Selected: " & pp_Calendar1.SelectedDate.ToShortDateString()
    End Sub
```

```
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Second Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Label ID="Label1" Runat="server"></asp:Label>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
        TextBox pp_Textbox1;
        Calendar pp_Calendar1;

        pp_Textbox1 = (TextBox)PreviousPage.FindControl("Textbox1");
        pp_Calendar1 = (Calendar)PreviousPage.FindControl("Calendar1");

        Label1.Text = "Hello " + pp_Textbox1.Text + "<br />" + "Date Selected: " +
           pp_Calendar1.SelectedDate.ToShortDateString();
    }
</script>
```

You have a couple of ways of getting at the values of the controls that are exposed from `Page1.aspx` from the second page. The first option is displayed in Listing 1-10. To get at a particular control's value that is carried over from the previous page, you simply create an instance of that control type and populate this instance using the `FindControl()` method from the `PreviousPage` property. The `String` value assigned to the `FindControl()` method is the `Id` value, which is used for the server control from the previous page. After this is assigned, you can work with the server control and its carried-over values just as if it had originally resided on the current page. You can see from the example that you can extract the `Text` and `SelectedDate` properties from the controls without any problem.

Another way of exposing the control values from the first page (`Page1.aspx`) is to create a `Property` for the control. This is shown in Listing 1-11.

Listing 1-11: Exposing the values of the control from a Property

VB

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

Continued

```
<script runat="server">
   Public ReadOnly Property pp_TextBox1() As TextBox
       Get
           Return TextBox1
       End Get
   End Property

   Public ReadOnly Property pp_Calendar1() As Calendar
       Get
           Return Calendar1
       End Get
   End Property

   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
       Label1.Text = "Hello " & TextBox1.Text & "<br />" & _
           "Date Selected: " & Calendar1.SelectedDate.ToShortDateString()
   End Sub
</script>
```

C#

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    public TextBox pp_TextBox1
    {
        get
        {
            return TextBox1;
        }
    }

    public Calendar pp_Calendar1
    {
        get
        {
            return Calendar1;
        }
    }

    protected void Button1_Click (object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + TextBox1.Text + "<br />" +
            "Date Selected: " + Calendar1.SelectedDate.ToShortDateString();
    }
</script>
```

Now that these properties are exposed on the posting page, the second page (`Page2.aspx`) can more easily work with the server control properties that are exposed from the first page. Listing 1-12 shows you how `Page2.aspx` works with these exposed properties.

Listing 1-12: Consuming the exposed properties from the first page

VB

```
<%@ Page Language="VB" %>
<%@ PreviousPageType VirtualPath="Page1.aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "Hello " & PreviousPage.pp_Textbox1.Text & "<br />" & _
            "Date Selected: " & _
            PreviousPage.pp_Calendar1.SelectedDate.ToShortDateString()
    End Sub
</script>
```

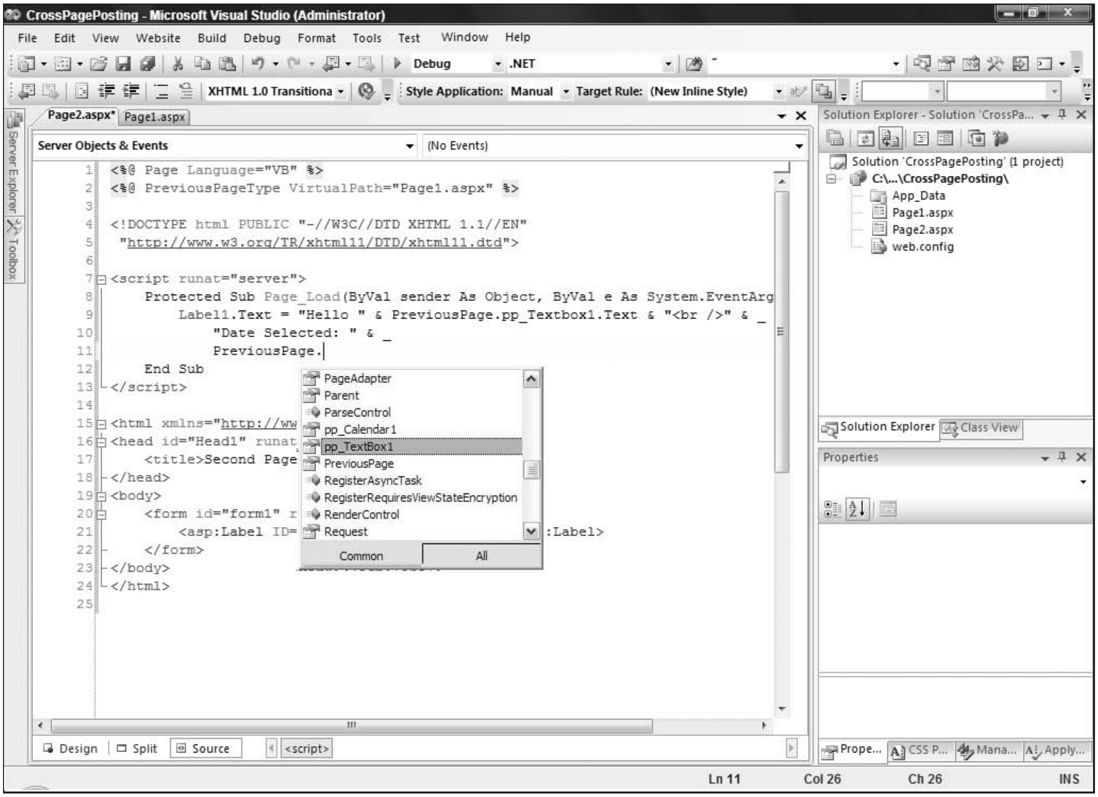
C#

```
<%@ Page Language="C#" %>
<%@ PreviousPageType VirtualPath="Page1.aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + PreviousPage.pp_TextBox1.Text + "<br />" +
           "Date Selected: " +
           PreviousPage.pp_Calendar1.SelectedDate.ToShortDateString();
    }
</script>
```

In order to be able to work with the properties that `Page1.aspx` exposes, you have to strongly type the `PreviousPage` property to `Page1.aspx`. To do this, you use the `PreviousPageType` directive. This new directive allows you to specifically point to `Page1.aspx` with the use of the `VirtualPath` attribute. When that is in place, notice that you can see the properties that `Page1.aspx` exposes through IntelliSense from the `PreviousPage` property. This is illustrated in Figure 1-7.

As you can see, working with cross-page posting is straightforward. Notice that, when you are cross posting from one page to another, you are not restricted to working only with the postback on the second page. In fact, you can still create methods on `Page1.aspx` that work with the postback before moving onto `Page2.aspx`. To do this, you simply add an `OnClick` event for the button in `Page1.aspx` and a method. You also assign a value for the `PostBackUrl` property. You can then work with the postback on `Page1.aspx` and then again on `Page2.aspx`.

What happens if someone requests `Page2.aspx` before she works her way through `Page1.aspx`? It is actually quite easy to determine if the request is coming from `Page1.aspx` or if someone just hit `Page2.aspx` directly. You can work with the request through the use of the `IsCrossPagePostBack` property that is quite similar to the `IsPostBack` property from ASP.NET 1.0/1.1. The `IsCrossPagePostBack` property enables you to check whether the request is from `Page1.aspx`. Listing 1-13 shows an example of this.


CrossPagePosting - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Format Tools Test Window Help
Debug
.NET
XHTML 1.0 Transitiona
Style Application: Manual
Target Rule: (New Inline Style)
Page2.aspx*
Page1.aspx
Server Objects & Events
(No Events)
PageAdapter
Parent
ParseControl
pp_Calendar1
pp_TextBox1
PreviousPage
RegisterAsyncTask
RegisterRequiresViewStateEncryption
RenderControl
Request
Common
All
Design
Split
Source
<script>
Solution Explorer - Solution 'CrossPa...
Solution 'CrossPagePosting' (1 project)
C:\...\CrossPagePosting\
App_Data
Page1.aspx
Page2.aspx
web.config
Solution Explorer
Class View
Properties
Prope...
CSS P...
Mana...
Apply...
Ln 11
Col 26
Ch 26
INS
Server Explorer
Toolbox

```
<%@ Page Language="VB" %>
<%@ PreviousPageType VirtualPath="Page1.aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArg
        Label1.Text = "Hello " & PreviousPage.pp_Textbox1.Text & "<br />" & _
            "Date Selected: " & _
            PreviousPage.
    End Sub
</script>

<html xmlns="http://ww
<head id="Head1" runat
    <title>Second Page
</head>
<body>
    <form id="form1" r
        <asp:Label ID=                              :Label>
    </form>
</body>
</html>
```

Figure 1-7

Listing 1-13: Using the IsCrossPagePostBack property

VB

```
<%@ Page Language="VB" %>
<%@ PreviousPageType VirtualPath="Page1.aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
      If Not PreviousPage Is Nothing And PreviousPage.IsCrossPagePostBack Then
        Label1.Text = "Hello " & PreviousPage.pp_Textbox1.Text & "<br />" & _
            "Date Selected: " & _
            PreviousPage.pp_Calendar1.SelectedDate.ToShortDateString()
      Else
        Response.Redirect("Page1.aspx")
      End If
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" %>
<%@ PreviousPageType VirtualPath="Page1.aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
      if (PreviousPage != null && PreviousPage.IsCrossPagePostBack) {
        Label1.Text = "Hello " + PreviousPage.pp_TextBox1.Text + "<br />" +
           "Date Selected: " +
           PreviousPage.pp_Calendar1.SelectedDate.ToShortDateString();
      }
      else
      {
         Response.Redirect("Page1.aspx");
      }
    }
</script>
```

ASP.NET Application Folders

When you create ASP.NET applications, notice that ASP.NET 3.5 uses a file-based approach. When working with ASP.NET, you can add as many files and folders as you want within your application without recompiling each and every time a new file is added to the overall solution. ASP.NET 3.5 includes the capability to automatically precompile your ASP.NET applications dynamically.

ASP.NET 1.0/1.1 compiled everything in your solution into a DLL. This is no longer necessary because ASP.NET applications now have a defined folder structure. By using the ASP.NET defined folders, you can have your code automatically compiled for you, your application themes accessible throughout your application, and your globalization resources available whenever you need them. Look at each of these defined folders to see how they work. The first folder reviewed is the `\App_Code` folder.

\App_Code Folder

The `\App_Code` folder is meant to store your classes, `.wsdl` files, and typed datasets. Any of these items stored in this folder are then automatically available to all the pages within your solution. The nice thing about the `\App_Code` folder is that when you place something inside this folder, Visual Studio 2008 automatically detects this and compiles it if it is a class (`.vb` or `.cs`), automatically creates your XML Web service proxy class (from the `.wsdl` file), or automatically creates a typed dataset for you from your `.xsd` files. After the files are automatically compiled, these items are then instantaneously available to any of your ASP.NET pages that are in the same solution. Look at how to employ a simple class in your solution using the `\App_Code` folder.

The first step is to create an `\App_Code` folder. To do this, simply right-click the solution and choose Add ASP.NET Folder➪App_Code. Right away you will notice that Visual Studio 2008 treats this folder differently than the other folders in your solution. The `\App_Code` folder is shown in a different color (gray) with a document pictured next to the folder icon. See Figure 1-8.

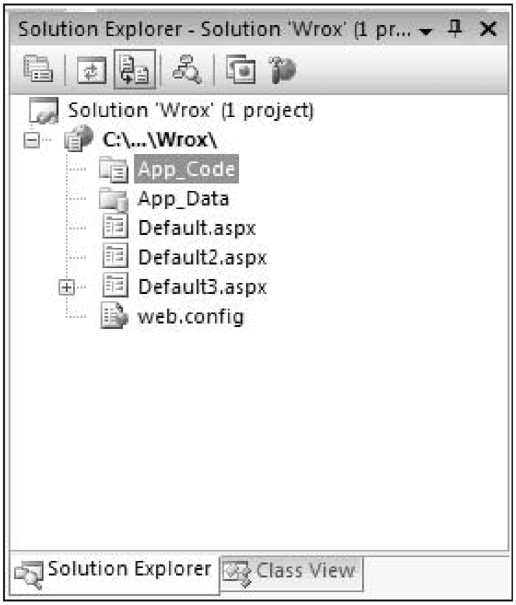

Solution Explorer - Solution 'Wrox' (1 pr...
Solution 'Wrox' (1 project)
C:\...\Wrox\
App_Code
App_Data
Default.aspx
Default2.aspx
Default3.aspx
web.config
Solution Explorer
Class View

Figure 1-8

After the `\App_Code` folder is in place, right-click the folder and select Add New Item. The Add New Item dialog that appears gives you a few options for the types of files that you can place within this folder. The available options include an AJAX-enabled WCF Service, a Class file, a LINQ to SQL Class, a Text file, a DataSet, a Report, and a Class Diagram if you are using Visual Studio 2008. Visual Web Developer 2008 Express Edition offers only the Class file, Text file, and DataSet file. For the first example, select the file of type Class and name the class `Calculator.vb` or `Calculator.cs`. Listing 1-14 shows how the `Calculator` class should appear.

Listing 1-14: The Calculator class

VB

```
Imports Microsoft.VisualBasic

Public Class Calculator
    Public Function Add(ByVal a As Integer, ByVal b As Integer) As Integer
        Return (a + b)
    End Function
End Class
```

C#

```
using System;

public class Calculator
```

```
{
    public int Add(int a, int b)
        {
            return (a + b);
    }
}
```

What's next? Just save this file, and it is now available to use in any pages that are in your solution. To see this in action, create a simple `.aspx` page that has just a single Label server control. Listing 1-15 shows you the code to place within the `Page_Load` event to make this new class available to the page.

Listing 1-15: An .aspx page that uses the Calculator class

VB

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim myCalc As New Calculator
        Label1.Text = myCalc.Add(12, 12)
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
        Calculator myCalc = new Calculator();
        Label1.Text = myCalc.Add(12, 12).ToString();
    }
</script>
```

When you run this `.aspx` page, notice that it utilizes the `Calculator` class without any problem, with no need to compile the class before use. In fact, right after saving the `Calculator` class in your solution or moving the class to the `\App_Code` folder, you also instantaneously receive IntelliSense capability on the methods that the class exposes (as illustrated in Figure 1-9).

To see how Visual Studio 2008 works with the `\App_Code` folder, open the `Calculator` class again in the IDE and add a `Subtract` method. Your class should now appear as shown in Listing 1-16.

Listing 1-16: Adding a Subtract method to the Calculator class

VB

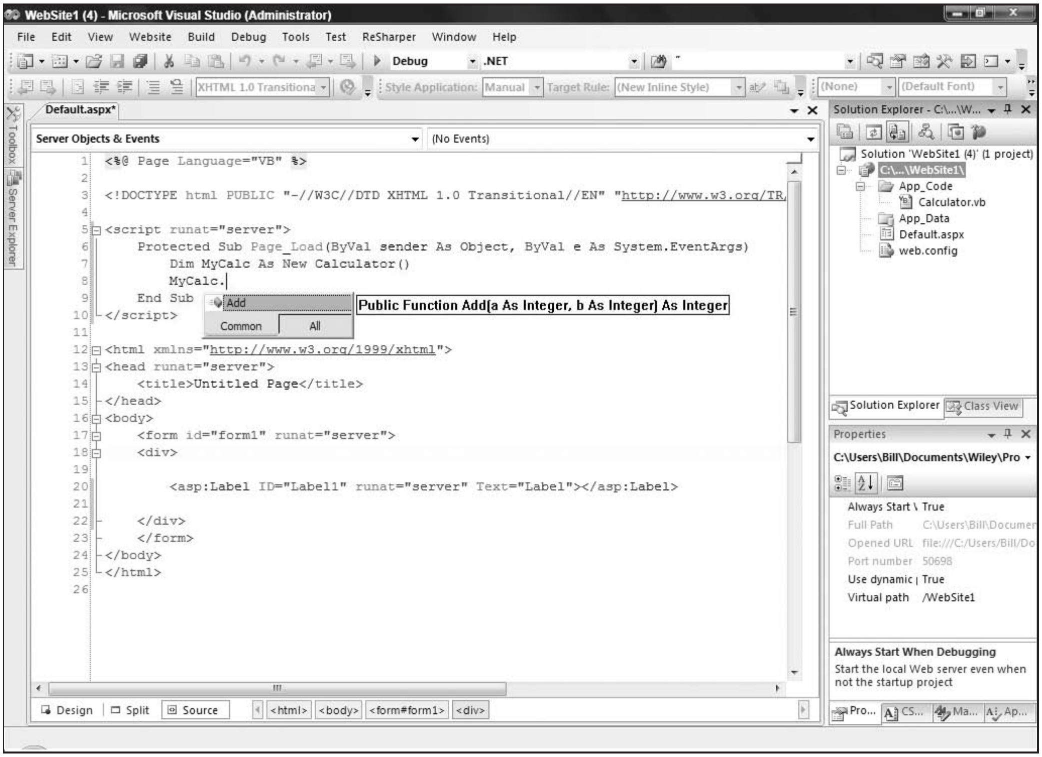
```
Imports Microsoft.VisualBasic
```

Continued

```
Public Class Calculator
    Public Function Add(ByVal a As Integer, ByVal b As Integer) As Integer
        Return (a + b)
    End Function

    Public Function Subtract(ByVal a As Integer, ByVal b As Integer) As Integer
        Return (a - b)
    End Function
End Class
```

C#

```
using System;

public class Calculator
{
    public int Add(int a, int b)
    {
            return (a + b);
    }

    public int Subtract(int a, int b)
    {
            return (a - b);
    }
}
```


WebSite1 (4) - Microsoft Visual Studio (Administrator)
File
Edit
View
Website
Build
Debug
Tools
Test
ReSharper
Window
Help
Debug
.NET
XHTML 1.0 Transitiona
Style Application:
Manual
Target Rule:
(New Inline Style)
(None)
(Default Font)
Toolbox
Server Explorer
Default.aspx*
Server Objects & Events
(No Events)
Add
Common
All
Solution Explorer - C:\...\W...
Solution 'WebSite1 (4)' (1 project)
C:\...\WebSite1\
App_Code
Calculator.vb
App_Data
Default.aspx
web.config
Solution Explorer
Class View
Properties
Design
Split
Source
<html>
<body>
<form#form1>
<div>
Pro...
CS...
Ma...
Ap...

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim MyCalc As New Calculator()
        MyCalc.
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>

    </div>
    </form>
</body>
</html>
```

Public Function Add(a As Integer, b As Integer) As Integer

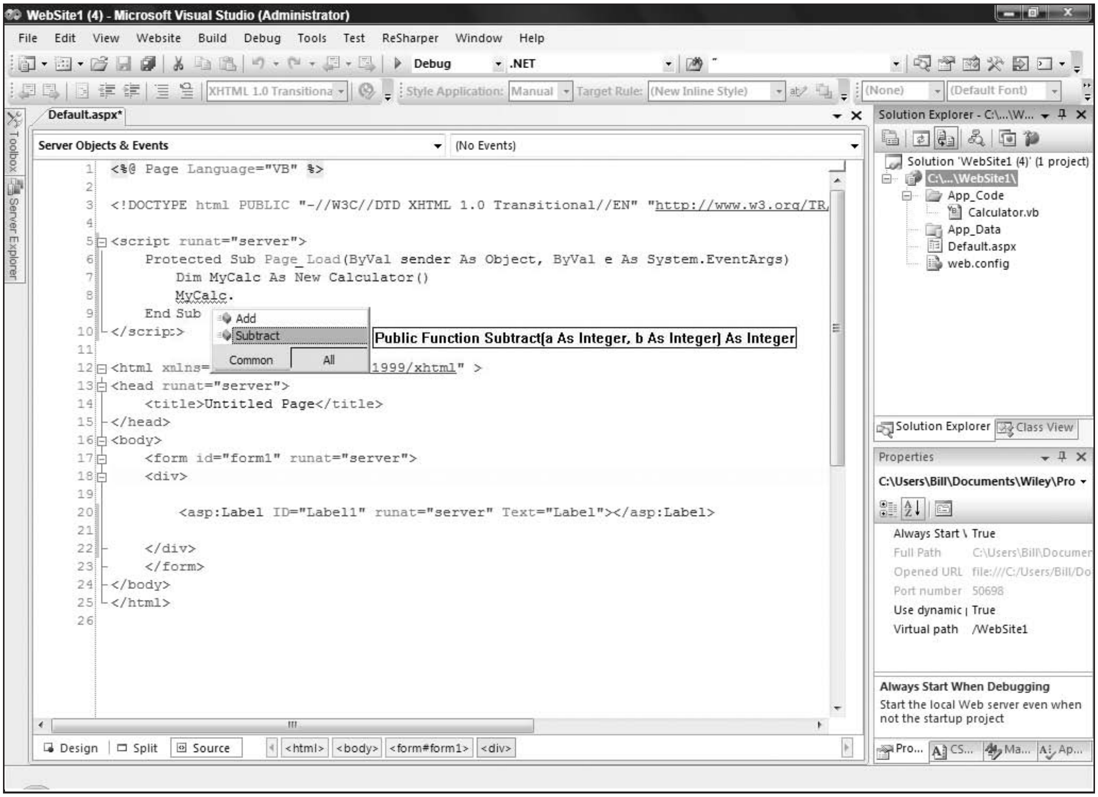
C:\Users\Bill\Documents\Wiley\Pro

Always Start \	True
Full Path	C:\Users\Bill\Documen
Opened URL	file:///C:/Users/Bill/Do
Port number	50698
Use dynamic	True
Virtual path	/WebSite1

Always Start When Debugging
Start the local Web server even when not the startup project

Figure 1-9

After you have added the `Subtract` method to the `Calculator` class, save the file and go back to your `.aspx` page. Notice that the class has been recompiled by the IDE, and the new method is now available to your page. You see this directly in IntelliSense. Figure 1-10 shows this in action.


WebSite1 (4) - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Tools Test ReSharper Window Help
Debug
.NET
XHTML 1.0 Transitiona
Style Application: Manual
Target Rule: (New Inline Style)
(None)
(Default Font)
Default.aspx*
Toolbox
Server Explorer
Server Objects & Events
(No Events)
Add
Subtract
Common
All
Design
Split
Source
<html> <body> <form#form1> <div>
Solution Explorer - C:\...\W...
Solution 'WebSite1 (4)' (1 project)
C:\...\WebSite1\
App_Code
Calculator.vb
App_Data
Default.aspx
web.config
Solution Explorer
Class View
Properties
C:\Users\Bill\Documents\Wiley\Pro
Pro...
CS...
Ma...
Ap...

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim MyCalc As New Calculator()
        MyCalc.
    End Sub
</script>

<html xmlns=                    1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>

    </div>
    </form>
</body>
</html>
```

Public Function Subtract(a As Integer, b As Integer) As Integer

Always Start \ True
Full Path C:\Users\Bill\Documen
Opened URL file:///C:/Users/Bill/Do
Port number 50698
Use dynamic | True
Virtual path /WebSite1

Always Start When Debugging
Start the local Web server even when not the startup project

Figure 1-10

Everything placed in the `\App_Code` folder is compiled into a single assembly. The class files placed within the `\App_Code` folder are not required to use a specific language. This means that even if all the pages of the solution are written in Visual Basic 2008, the `Calculator` class in the `\App_Code` folder of the solution can be built in C# (`Calculator.cs`).

Because all the classes contained in this folder are built into a single assembly, you *cannot* have classes of different languages sitting in the root `\App_Code` folder, as in the following example:

```
\App_Code
   Calculator.cs
   AdvancedMath.vb
```

Having two classes made up of different languages in the `\App_Code` folder (as shown here) causes an error to be thrown. It is impossible for the assigned compiler to work with two different languages. Therefore, in order to be able to work with multiple languages in your `\App_Code` folder, you must make some changes to the folder structure and to the `web.config` file.

The first step is to add two new subfolders to the `\App_Code` folder — a `\VB` folder and a `\CS` folder. This gives you the following folder structure:

```
\App_Code
   \VB
      Add.vb
   \CS
      Subtract.cs
```

This still will not correctly compile these class files into separate assemblies, at least not until you make some additions to the `web.config` file. Most likely, you do not have a `web.config` file in your solution at this moment, so add one through the Solution Explorer. After it is added, change the `<compilation>` node so that it is structured as shown in Listing 1-17.

Listing 1-17: Structuring the web.config file so that classes in the \App_Code folder can use different languages

```
<compilation>
   <codeSubDirectories>
      <add directoryName="VB"></add>
      <add directoryName="CS"></add>
   </codeSubDirectories>
</compilation>
```

Now that this is in place in your `web.config` file, you can work with each of the classes in your ASP.NET pages. In addition, any C# class placed in the `CS` folder is now automatically compiled just like any of the classes placed in the `VB` folder. Because you can add these directories in the `web.config` file, you are not required to name them `VB` and `CS` as we did; you can use whatever name tickles your fancy.

\App_Data Folder

The `\App_Data` folder holds the data stores utilized by the application. It is a good spot to centrally store all the data stores your application might use. The `\App_Data` folder can contain Microsoft SQL Express files (`.mdf` files), Microsoft Access files (`.mdb` files), XML files, and more.

The user account utilized by your application will have read and write access to any of the files contained within the `\App_Data` folder. By default, this is the ASPNET account. Another reason for storing all your data files in this folder is that much of the ASP.NET system — from the membership and role management systems to the GUI tools, such as the ASP.NET MMC snap-in and ASP.NET Web Site Administration Tool — is built to work with the `\App_Data` folder.

\App_Themes Folder

Themes are a new way of providing a common look-and-feel to your site across every page. You implement a theme by using a `.skin` file, CSS files, and images used by the server controls of your site. All these elements can make a *theme*, which is then stored in the `\App_Themes` folder of your solution. By storing these elements within the `\App_Themes` folder, you ensure that all the pages within the solution can take advantage of the theme and easily apply its elements to the controls and markup of the page. Themes are discussed in great detail in Chapter 6 of this book.

\App_GlobalResources Folder

Resource files are string tables that can serve as data dictionaries for your applications when these applications require changes to content based on things such as changes in culture. You can add Assembly Resource Files (`.resx`) to the `\App_GlobalResources` folder, and they are dynamically compiled and made part of the solution for use by all your `.aspx` pages in the application. When using ASP.NET 1.0/1.1, you had to use the `resgen.exe` tool and had to compile your resource files to a `.dll` or `.exe` for use within your solution. It is considerably easier to deal with resource files in ASP.NET 3.5. Simply placing your application-wide resources in this folder makes them instantly accessible. Localization is covered in detail in Chapter 31.

\App_LocalResources

Even if you are not interested in constructing application-wide resources using the `\App_GlobalResources` folder, you may want resources that can be used for a single `.aspx` page. You can do this very simply by using the `\App_LocalResources` folder.

You can add resource files that are page-specific to the `\App_LocalResources` folder by constructing the name of the `.resx` file in the following manner:

- `Default.aspx.resx`
- `Default.aspx.fi.resx`
- `Default.aspx.ja.resx`
- `Default.aspx.en-gb.resx`

Now, the resource declarations used on the `Default.aspx` page are retrieved from the appropriate file in the `\App_LocalResources` folder. By default, the `Default.aspx.resx` resource file is used if another match is not found. If the client is using a culture specification of `fi-FI` (Finnish), however, the `Default.aspx.fi.resx` file is used instead. Localization of local resources is covered in detail in Chapter 30.

\App_WebReferences

The `\App_WebReferences`folder is a new name for the previous `Web References` folder used in previous versions of ASP.NET. Now you can use the `\App_WebReferences` folder and have automatic access to the remote Web services referenced from your application. Web services in ASP.NET are covered in Chapter 30.

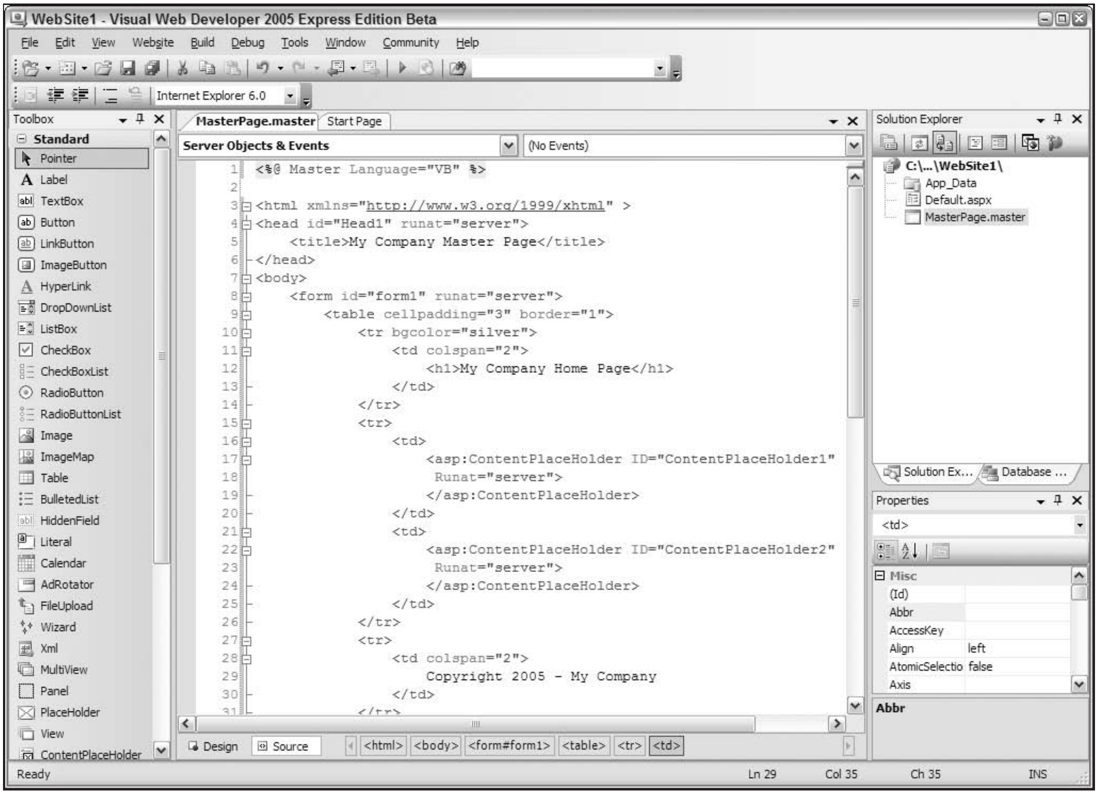
\App_Browsers

The `\App_Browsers` folder holds `.browser` files, which are XML files used to identity the browsers making requests to the application and understanding the capabilities these browsers have. You can find a list of globally accessible `.browser` files at `C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG\Browsers`. In addition, if you want to change any part of these default browser definition files, just copy the appropriate `.browser` file from the `Browsers` folder to your application's `\App_Browsers` folder and change the definition.

Compilation

You already saw how Visual Studio 2008 compiles pieces of your application as you work with them (for instance, by placing a class in the `\App_Code` folder). The other parts of the application, such as the `.aspx` pages, can be compiled just as they were in earlier versions of ASP.NET by referencing the pages in the browser.

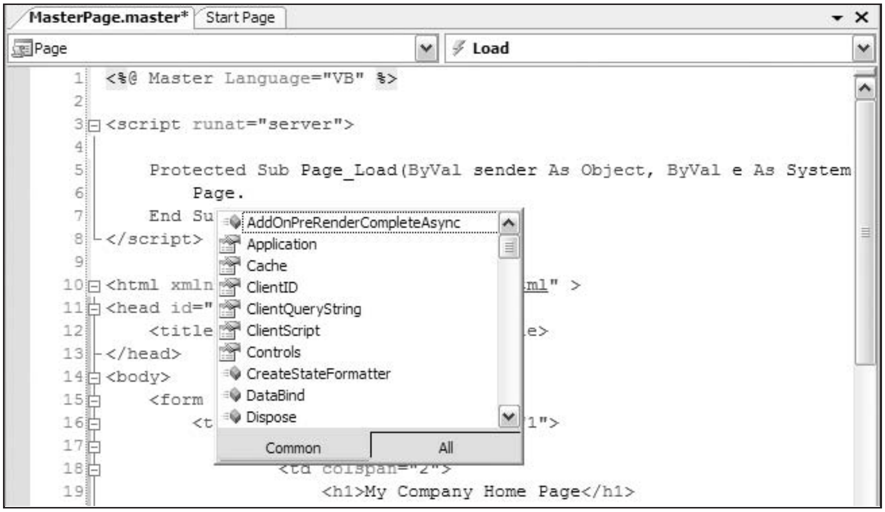
When an ASP.NET page is referenced in the browser for the first time, the request is passed to the ASP.NET parser that creates the class file in the language of the page. It is passed to the ASP.NET parser based on the file's extension (`.aspx`) because ASP.NET realizes that this file extension type is meant for its handling and processing. After the class file has been created, the class file is compiled into a DLL and then written to the disk of the Web server. At this point, the DLL is instantiated and processed, and an output is generated for the initial requester of the ASP.NET page. This is detailed in Figure 1-11.


WebSite1 - Visual Web Developer 2005 Express Edition Beta
File
Edit
View
Website
Build
Debug
Tools
Window
Community
Help
Internet Explorer 6.0
Toolbox
Standard
Pointer
Label
TextBox
Button
LinkButton
ImageButton
HyperLink
DropDownList
ListBox
CheckBox
CheckBoxList
RadioButton
RadioButtonList
Image
ImageMap
Table
BulletedList
HiddenField
Literal
Calendar
AdRotator
FileUpload
Wizard
Xml
MultiView
Panel
PlaceHolder
View
ContentPlaceHolder
MasterPage.master
Start Page
Server Objects & Events
(No Events)
Design
Source
<html>
<body>
<form#form1>
<table>
<tr>
<td>
Solution Explorer
C:\...\WebSite1\
App_Data
Default.aspx
MasterPage.master
Solution Ex...
Database ...
Properties
<td>
Misc
(Id)
Abbr
AccessKey
Align
left
AtomicSelectio
false
Axis
Abbr
Ready
Ln 29
Col 35
Ch 35
INS

```
<%@ Master Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>My Company Master Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <table cellpadding="3" border="1">
            <tr bgcolor="silver">
                <td colspan="2">
                    <h1>My Company Home Page</h1>
                </td>
            </tr>
            <tr>
                <td>
                    <asp:ContentPlaceHolder ID="ContentPlaceHolder1"
                     Runat="server">
                    </asp:ContentPlaceHolder>
                </td>
                <td>
                    <asp:ContentPlaceHolder ID="ContentPlaceHolder2"
                     Runat="server">
                    </asp:ContentPlaceHolder>
                </td>
            </tr>
            <tr>
                <td colspan="2">
                    Copyright 2005 - My Company
                </td>
            </tr>
```

Figure 1-11

On the next request, great things happen. Instead of going through the entire process again for the second and respective requests, the request simply causes an instantiation of the already-created DLL, which sends out a response to the requester. This is illustrated in Figure 1-12.



```
<%@ Master Language="VB" %>

<script runat="server">

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System
        Page.
    End Su
</script>

<html xmln                                   ml" >
<head id="
    <title                                   e>
</head>
<body>
    <form
        <t                                   1">

                <td colspan="2">
                    <h1>My Company Home Page</h1>
```

Figure 1-12

Because of the mechanics of this process, if you made changes to your `.aspx` code-behind pages, you found it necessary to recompile your application. This was quite a pain if you had a larger site and did not want your end users to experience the extreme lag that occurs when an `.aspx` page is referenced for the first time after compilation. Many developers, consequently, began to develop their own tools that automatically go out and hit every single page within their application to remove this first-time lag hit from the end user's browsing experience.

ASP.NET 3.5 provides a few ways to precompile your entire application with a single command that you can issue through a command line. One type of compilation is referred to as *in-place precompilation*. In order to precompile your entire ASP.NET application, you must use the `aspnet_compiler.exe` tool that now comes with ASP.NET 3.5. You navigate to the tool using the Command window. Open the Command window and navigate to `C:\Windows\Microsoft.NET\Framework\v2.0.50727\`. When you are there, you can work with the `aspnet_compiler` tool. You can also get to this tool directly by pulling up the Visual Studio 2008 Command Prompt. Choose Start➪All Programs➪Microsoft Visual Studio 2008➪Visual Studio Tools➪Visual Studio 2008 Command Prompt.

After you get the command prompt, you use the aspnet_compiler.exe tool to perform an in-place precompilation using the following command:

```
aspnet_compiler -p "C:\Inetpub\wwwroot\WROX" -v none
```

You then get a message stating that the precompilation is successful. The other great thing about this precompilation capability is that you can also use it to find errors on any of the ASP.NET pages in your application. Because it hits each and every page, if one of the pages contains an error that won't be triggered until runtime, you get notification of the error immediately as you employ this precompilation method.

The next precompilation option is commonly referred to as *precompilation for deployment*. This is an outstanding capability of ASP.NET that enables you to compile your application down to some DLLs, which can then be deployed to customers, partners, or elsewhere for your own use. Not only are minimal steps required to do this, but also after your application is compiled, you simply have to move around the DLL and some placeholder files for the site to work. This means that your Web site code is completely removed and placed in the DLL when deployed.

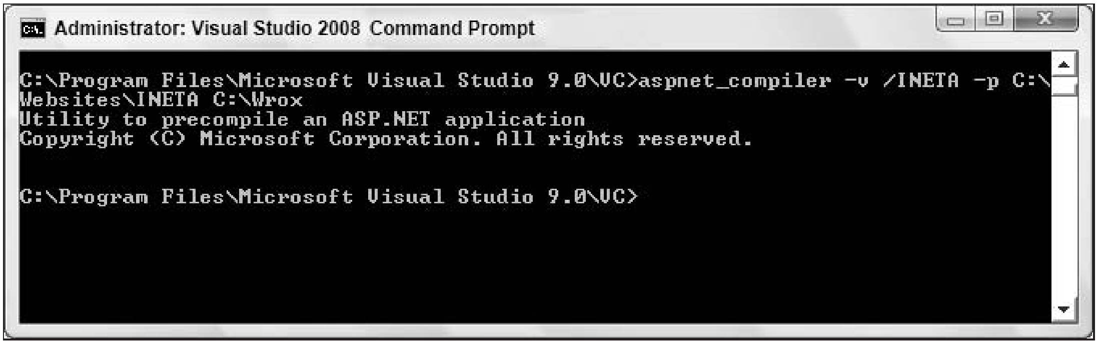
However, before you take these precompilation steps, create a folder in your root drive called, for example, `Wrox`. This folder is the one to which you will direct the compiler output. When it is in place, you can return to the compiler tool and give the following command:

```
aspnet_compiler -v [Application Name] -p [Physical Location] [Target]
```

Therefore, if you have an application called `INETA` located at `C:\Websites\INETA`, you use the following commands:

```
aspnet_compiler -v /INETA -p C:\Websites\INETA C:\Wrox
```

Press the Enter key, and the compiler either tells you that it has a problem with one of the command parameters or that it was successful (shown in Figure 1-13). If it was successful, you can see the output placed in the target directory.



```
C:\Program Files\Microsoft Visual Studio 9.0\VC>aspnet_compiler -v /INETA -p C:\
Websites\INETA C:\Wrox
Utility to precompile an ASP.NET application
Copyright (C) Microsoft Corporation. All rights reserved.


C:\Program Files\Microsoft Visual Studio 9.0\VC>
```

Figure 1-13

In the example just shown, `-v` is a command for the virtual path of the application, which is provided by using `/INETA`. The next command is `-p`, which is pointing to the physical path of the application. In this case, it is `C:\Websites\INETA`. Finally, the last bit, `C:\Wrox`, is the location of the compiler output. The following table describes some of the possible commands for the `aspnet_compiler.exe` tool.

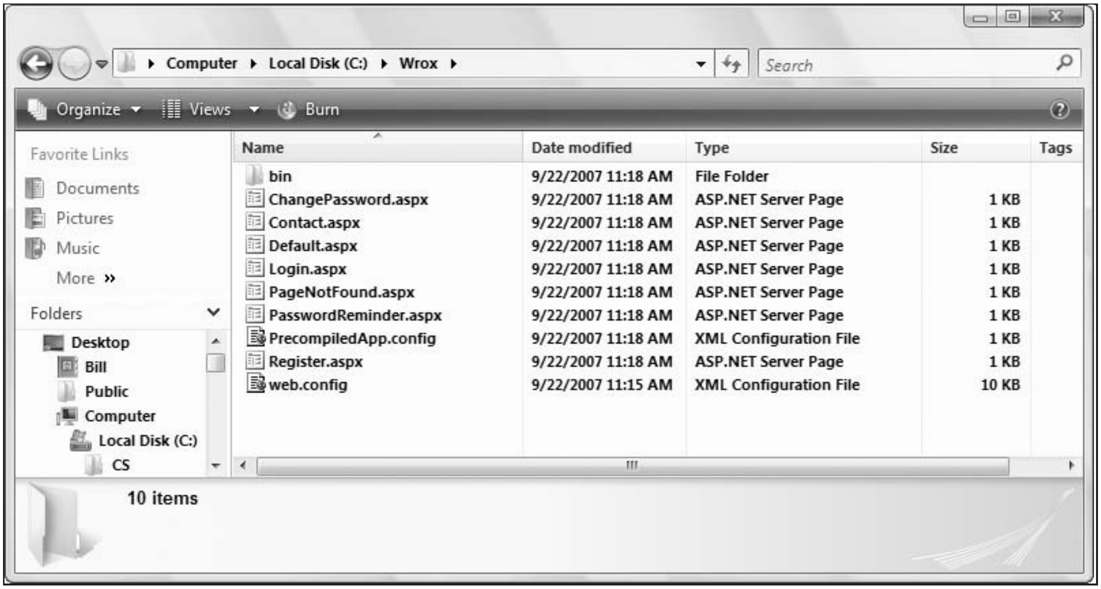
Command	Description
`-m`	Specifies the full IIS metabase path of the application. If you use the `-m` command, you cannot use the `-v` or `-p` command.
`-v`	Specifies the virtual path of the application to be compiled. If you also use the `-p` command, the physical path is used to find the location of the application.

Command	Description
`-p`	Specifies the physical path of the application to be compiled. If this is not specified, the IIS metabase is used to find the application.
`-u`	If this command is utilized, it specifies that the application is updatable.
`-f`	Specifies to overwrite the target directory if it already exists.
`-d`	Specifies that the debug information should be excluded from the compilation process.
`[targetDir]`	Specifies the target directory where the compiled files should be placed. If this is not specified, the output files are placed in the application directory.

After compiling the application, you can go to `C:\Wrox` to see the output. Here, you see all the files and the file structures that were in the original application. However, if you look at the content of one of the files, notice that the file is simply a placeholder. In the actual file, you find the following comment:

```
This is a marker file generated by the precompilation tool
and should not be deleted!
```

In fact, you find a `Code.dll` file in the `bin` folder where all the page code is located. Because it is in a DLL file, it provides great code obfuscation as well. From here on, all you do is move these files to another server using FTP or Windows Explorer, and you can run the entire Web application from these files. When you have an update to the application, you simply provide a new set of compiled files. A sample output is displayed in Figure 1-14.



Name	Date modified	Type	Size	Tags
bin	9/22/2007 11:18 AM	File Folder		
ChangePassword.aspx	9/22/2007 11:18 AM	ASP.NET Server Page	1 KB	
Contact.aspx	9/22/2007 11:18 AM	ASP.NET Server Page	1 KB	
Default.aspx	9/22/2007 11:18 AM	ASP.NET Server Page	1 KB	
Login.aspx	9/22/2007 11:18 AM	ASP.NET Server Page	1 KB	
PageNotFound.aspx	9/22/2007 11:18 AM	ASP.NET Server Page	1 KB	
PasswordReminder.aspx	9/22/2007 11:18 AM	ASP.NET Server Page	1 KB	
PrecompiledApp.config	9/22/2007 11:18 AM	XML Configuration File	1 KB	
Register.aspx	9/22/2007 11:18 AM	ASP.NET Server Page	1 KB	
web.config	9/22/2007 11:15 AM	XML Configuration File	10 KB	

Figure 1-14

Note that this compilation process does not compile *every* type of Web file. In fact, it compiles only the ASP.NET-specific file types and leaves out of the compilation process the following types of files:

- ❑ HTML files
- ❑ XML files
- ❑ XSD files
- ❑ `web.config` files
- ❑ Text files

You cannot do much to get around this, except in the case of the HTML files and the text files. For these file types, just change the file extensions of these file types to `.aspx`; they are then compiled into the `Code.dll` like all the other ASP.NET files.

Build Providers

As you review the various ASP.NET folders, note that one of the more interesting folders is the \App_Code folder. You can simply drop code files, XSD files, and even WSDL files directly into the folder for automatic compilation. When you drop a class file into the \App_Code folder, the class can automatically be utilized by a running application. In the early days of ASP.NET, if you wanted to deploy a custom component, you had to precompile the component before being able to utilize it within your application. Now ASP.NET simply takes care of all the work that you once had to do. You do not need to perform any compilation routine.

Which file types are compiled in the App_Code folder? As with most things in ASP.NET, this is determined through settings applied in a configuration file. Listing 1-18 shows a snippet of configuration code taken from the master `Web.config` file found in ASP.NET 3.5.

Listing 1-18: Reviewing the list of build providers

```
<compilation>
   <buildProviders>
      <add extension=".aspx" type="System.Web.Compilation.PageBuildProvider" />
      <add extension=".ascx"
       type="System.Web.Compilation.UserControlBuildProvider" />
      <add extension=".master"
       type="System.Web.Compilation.MasterPageBuildProvider" />
      <add extension=".asmx"
       type="System.Web.Compilation.WebServiceBuildProvider" />
      <add extension=".ashx"
       type="System.Web.Compilation.WebHandlerBuildProvider" />
      <add extension=".soap"
       type="System.Web.Compilation.WebServiceBuildProvider" />
      <add extension=".resx" type="System.Web.Compilation.ResXBuildProvider" />
      <add extension=".resources"
       type="System.Web.Compilation.ResourcesBuildProvider" />
      <add extension=".wsdl" type="System.Web.Compilation.WsdlBuildProvider" />
      <add extension=".xsd" type="System.Web.Compilation.XsdBuildProvider" />
      <add extension=".js" type="System.Web.Compilation.ForceCopyBuildProvider" />
```

```
        <add extension=".lic"
         type="System.Web.Compilation.IgnoreFileBuildProvider" />
        <add extension=".licx"
         type="System.Web.Compilation.IgnoreFileBuildProvider" />
        <add extension=".exclude"
         type="System.Web.Compilation.IgnoreFileBuildProvider" />
        <add extension=".refresh"
         type="System.Web.Compilation.IgnoreFileBuildProvider" />
     </buildProviders>
  </compilation>
```

This section contains a list of build providers that can be used by two entities in your development cycle. The build provider is first used is during development when you are building your solution in Visual Studio 2008. For instance, placing a `.wsdl` file in the App_Code folder during development in Visual Studio causes the IDE to give you automatic access to the dynamically compiled proxy class that comes from this `.wsdl` file. The other entity that uses the build providers is ASP.NET itself. As stated, simply dragging and dropping a `.wsdl` file in the App_Code folder of a deployed application automatically gives the ASP.NET application access to the created proxy class.

A build provider is simply a class that inherits from `System.Web.Compilation.BuildProvider`. The `<buildProviders>` section in the `Web.config` allows you to list the build provider classes that will be utilized. The capability to dynamically compile any WSDL file is defined by the following line in the configuration file.

```
<add extension=".wsdl" type="System.Web.Compilation.WsdlBuildProvider" />
```

This means that any file utilizing the `.wsdl` file extension is compiled using the `WsdlBuildProvider`, a class that inherits from `BuildProvider`. Microsoft provides a set number of build providers out of the box for you to use. As you can see from the set in Listing 1-18, a number of providers are available in addition to the `WsdlBuildProvider`, including providers such as the `XsdBuildProvider`, `PageBuildProvider`, `UserControlBuildProvider`, `MasterPageBuildProvider`, and more. Just by looking at the names of some of these providers you can pretty much understand what they are about. The next section, however, reviews some other providers whose names might not ring a bell right away.

Using the Built-in Build Providers

Two of the providers that this section covers are the `ForceCopyBuildProvider` and the `IgnoreFileBuildProvider`, both of which are included in the default list of providers.

The `ForceCopyBuildProvider` is basically a provider that copies only those files for deployment that use the defined extension. (These files are not included in the compilation process.) An extension that utilizes the `ForceCopyBuildProvider` is shown in the predefined list in Listing 1-18. This is the `.js` file type (a JavaScript file extension). Any `.js` files are simply copied and not included in the compilation process (which makes sense for JavaScript files). You can add other file types that you want to be a part of this copy process with the command shown here:

```
<add extension=".chm" type="System.Web.Compilation.ForceCopyBuildProvider" />
```

In addition to the `ForceCopyBuildProvider`, you should also be aware of the `IgnoreFileBuildProvider` class. This provider causes the defined file type to be ignored in the deployment or compilation process. This means that any file type defined with `IgnoreFileBuildProvider` is simply ignored. Visual Studio will not copy, compile, or deploy any file of that type. So, if you are including Visio diagrams in your project, you can simply add the following `<add>` element to the `web.config` file to have this file type ignored. An example is presented here:

```
<add extension=".vsd" type="System.Web.Compilation.IgnoreFileBuildProvider" />
```

With this in place, all `.vsd` files are ignored.

Using Your Own Build Providers

In addition to using the predefined build providers out of the box, you can also take this build provider stuff one-step further and construct your own custom build providers to use within your applications.

For example, suppose you wanted to construct a `Car` class dynamically based upon settings applied in a custom `.car` file that you have defined. You might do this because you are using this `.car` definition file in multiple projects or many times within the same project. Using a build provider makes it simpler to define these multiple instances of the `Car` class.

An example of the `.car` file type is presented in Listing 1-19.

Listing 1-19: An example of a .car file

```
<?xml version="1.0" encoding="utf-8" ?>
<car name="EvjenCar">
  <color>Blue</color>
  <door>4</door>
  <speed>150</speed>
</car>
```

In the end, this XML declaration specifies the name of the class to compile as well as some values for various properties and a method. These elements make up the class. Now that you understand the structure of the `.car` file type, the next step is to construct the build provider. To accomplish this task, create a new Class Library project in the language of your choice within Visual Studio. Name the project `CarBuildProvider`. The `CarBuildProvider` contains a single class — `Car.vb` or `Car.cs`. This class inherits from the base class `BuildProvider` and overrides the `GenerateCode()` method of the `BuildProvider` class. This class is presented in Listing 1-20.

Listing 1-20: The CarBuildProvider

VB

```
Imports System.IO
Imports System.Web.Compilation
Imports System.Xml
Imports System.CodeDom

Public Class Car
    Inherits BuildProvider

    Public Overrides Sub GenerateCode(ByVal myAb As AssemblyBuilder)
        Dim carXmlDoc As XmlDocument = New XmlDocument()
        Using passedFile As Stream = Me.OpenStream()
            carXmlDoc.Load(passedFile)
        End Using

        Dim mainNode As XmlNode = carXmlDoc.SelectSingleNode("/car")
        Dim selectionMainNode As String = mainNode.Attributes("name").Value

        Dim colorNode As XmlNode = carXmlDoc.SelectSingleNode("/car/color")
        Dim selectionColorNode As String = colorNode.InnerText

        Dim doorNode As XmlNode = carXmlDoc.SelectSingleNode("/car/door")
        Dim selectionDoorNode As String = doorNode.InnerText

        Dim speedNode As XmlNode = carXmlDoc.SelectSingleNode("/car/speed")
        Dim selectionSpeedNode As String = speedNode.InnerText

        Dim ccu As CodeCompileUnit = New CodeCompileUnit()
        Dim cn As CodeNamespace = New CodeNamespace()
        Dim cmp1 As CodeMemberProperty = New CodeMemberProperty()
        Dim cmp2 As CodeMemberProperty = New CodeMemberProperty()
        Dim cmm1 As CodeMemberMethod = New CodeMemberMethod()

        cn.Imports.Add(New CodeNamespaceImport("System"))

        cmp1.Name = "Color"
        cmp1.Type = New CodeTypeReference(GetType(System.String))
        cmp1.Attributes = MemberAttributes.Public
        cmp1.GetStatements.Add(New CodeSnippetExpression("return """ & _
           selectionColorNode & """"))

        cmp2.Name = "Doors"
        cmp2.Type = New CodeTypeReference(GetType(System.Int32))
        cmp2.Attributes = MemberAttributes.Public
        cmp2.GetStatements.Add(New CodeSnippetExpression("return " & _
           selectionDoorNode))

        cmm1.Name = "Go"
        cmm1.ReturnType = New CodeTypeReference(GetType(System.Int32))
        cmm1.Attributes = MemberAttributes.Public
        cmm1.Statements.Add(New CodeSnippetExpression("return " & _
           selectionSpeedNode))

        Dim ctd As CodeTypeDeclaration = New CodeTypeDeclaration(selectionMainNode)
        ctd.Members.Add(cmp1)
        ctd.Members.Add(cmp2)
        ctd.Members.Add(cmm1)

        cn.Types.Add(ctd)
        ccu.Namespaces.Add(cn)

        myAb.AddCodeCompileUnit(Me, ccu)
    End Sub

End Class
```

Continued

C#

```
using System.IO;
using System.Web.Compilation;
using System.Xml;
using System.CodeDom;

namespace CarBuildProvider
{
    class Car : BuildProvider
    {
        public override void GenerateCode(AssemblyBuilder myAb)
        {
            XmlDocument carXmlDoc = new XmlDocument();

            using (Stream passedFile = OpenStream())
            {
                carXmlDoc.Load(passedFile);
            }
            XmlNode mainNode = carXmlDoc.SelectSingleNode("/car");
            string selectionMainNode = mainNode.Attributes["name"].Value;

            XmlNode colorNode = carXmlDoc.SelectSingleNode("/car/color");
            string selectionColorNode = colorNode.InnerText;

            XmlNode doorNode = carXmlDoc.SelectSingleNode("/car/door");
            string selectionDoorNode = doorNode.InnerText;

            XmlNode speedNode = carXmlDoc.SelectSingleNode("/car/speed");
            string selectionSpeedNode = speedNode.InnerText;

            CodeCompileUnit ccu = new CodeCompileUnit();
            CodeNamespace cn = new CodeNamespace();
            CodeMemberProperty cmp1 = new CodeMemberProperty();
            CodeMemberProperty cmp2 = new CodeMemberProperty();
            CodeMemberMethod cmm1 = new CodeMemberMethod();

            cn.Imports.Add(new CodeNamespaceImport("System"));

            cmp1.Name = "Color";
            cmp1.Type = new CodeTypeReference(typeof(string));
            cmp1.Attributes = MemberAttributes.Public;
            cmp1.GetStatements.Add(new CodeSnippetExpression("return \"" +
               selectionColorNode + "\""));

            cmp2.Name = "Doors";
            cmp2.Type = new CodeTypeReference(typeof(int));
            cmp2.Attributes = MemberAttributes.Public;
            cmp2.GetStatements.Add(new CodeSnippetExpression("return " +
               selectionDoorNode));

            cmm1.Name = "Go";
            cmm1.ReturnType = new CodeTypeReference(typeof(int));
            cmm1.Attributes = MemberAttributes.Public;
```

Continued

```
                cmm1.Statements.Add(new CodeSnippetExpression("return " +
                   selectionSpeedNode));

                CodeTypeDeclaration ctd = new CodeTypeDeclaration(selectionMainNode);
                ctd.Members.Add(cmp1);
                ctd.Members.Add(cmp2);
                ctd.Members.Add(cmm1);

                cn.Types.Add(ctd);
                ccu.Namespaces.Add(cn);

                myAb.AddCodeCompileUnit(this, ccu);
            }
        }
    }
```

As you look over the `GenerateCode()` method, you can see that it takes an instance of `AssemblyBuilder`. This `AssemblyBuilder` object is from the `System.Web.Compilation` namespace and, because of this, your Class Library project needs to have a reference to the `System.Web` assembly. With all the various objects used in this `Car` class, you also have to import in the following namespaces:

```
Imports System.IO
Imports System.Web.Compilation
Imports System.Xml
Imports System.CodeDom
```

When you have done this, one of the tasks remaining in the `GenerateCode()` method is loading the `.car` file. Because the .car file is using XML for its form, you are able to load the document easily using the `XmlDocument` object. From there, by using the CodeDom, you can create a class that contains two properties and a single method dynamically. The class that is generated is an abstract representation of what is defined in the provided `.car` file. On top of that, the name of the class is also dynamically driven from the value provided via the name attribute used in the main `<Car>` node of the `.car` file.

The `AssemblyBuilder` instance that is used as the input object then compiles the generated code along with everything else into an assembly.

What does it mean that your ASP.NET project has a reference to the `CarBuildProvider` assembly in its project? It means that you can create a `.car` file of your own definition and drop this file into the App_Code folder. The second you drop the file into the App_Code folder, you have instant programmatic access to the definition specified in the file.

To see this in action, you first need a reference to the build provider in either the server's `machine.config` or your application's `web.config` file. A reference is shown in Listing 1-21.

Listing 1-21: Making a reference to the build provider in the web.config file

```
<configuration>
 <system.web>
  <compilation debug="false">
   <buildProviders>
    <add extension=".car" type="CarBuildProvider.Car"/>
```

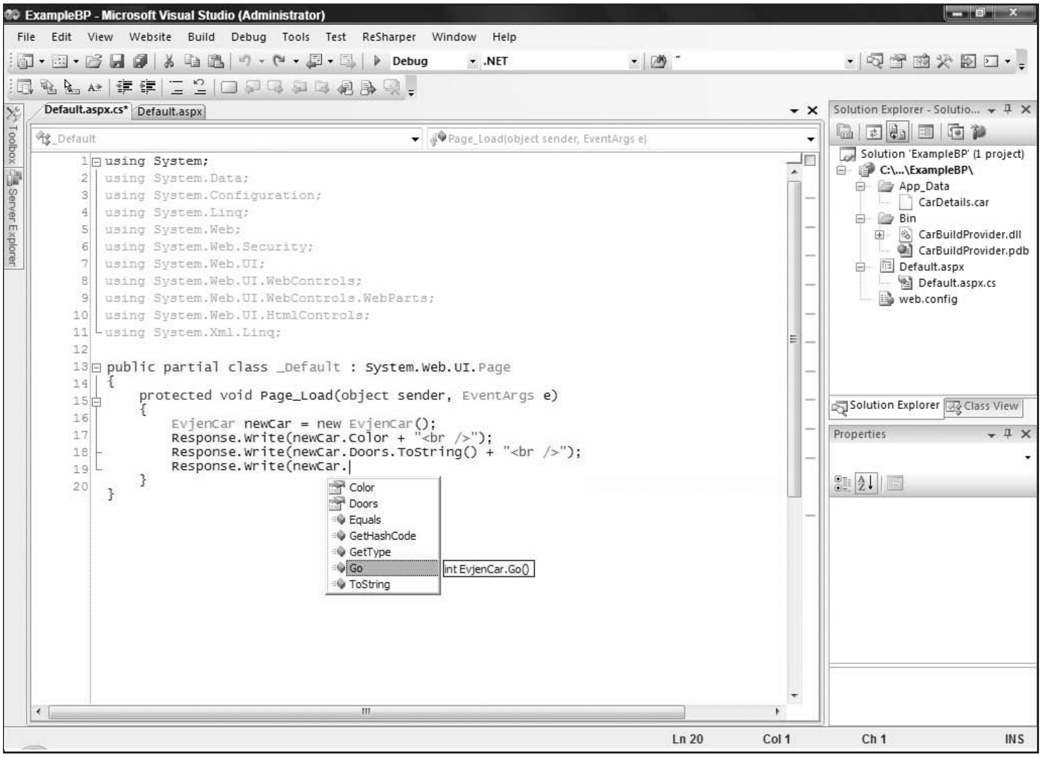
Continued

```
            </buildProviders>
        </compilation>
    </system.web>
</configuration>
```

The `<buildProviders>` element is a child element of the `<compilation>` element. The `<buildProviders>` element takes a couple of child elements to add or remove providers. In this case, because you want to add a reference to the custom `CarBuildProvider` object, you use the `<add>` element. The `<add>` element can take two possible attributes — `extension` and `type`. You must use both of these attributes. In `extension` attribute, you define the file extension that this build provider will be associated with. In this case, you use the `.car` file extension. This means that any file using this file extension is associated with the class defined in the `type` attribute. The `type` attribute then takes a reference to the `CarBuildProvider` class that you built — `CarBuildProvider.Car`.

With this reference in place, you can create the `.car` file that was shown earlier in Listing 1-19. Place the created .car file in the App_Code folder. You instantly have access to a dynamically generated class that comes from the definition provided via the file. For example, because I used `EvjenCar` as the value of the name attribute in the `<Car>` element, this will be the name of the class generated, and I will find this exact name in IntelliSense as I type in Visual Studio.

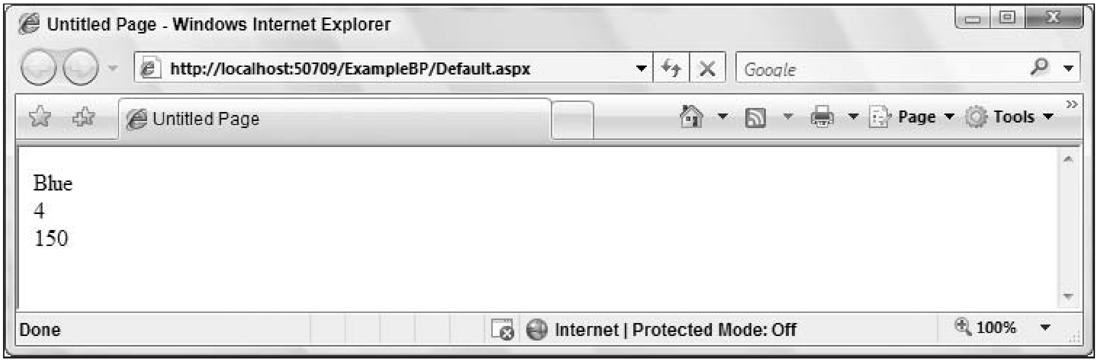
If you create an instance of the `EvjenCar` class, you also find that you have access to the properties and the method that this class exposes. This is shown in Figure 1-15.


ExampleBP - Microsoft Visual Studio (Administrator)
File
Edit
View
Website
Build
Debug
Tools
Test
ReSharper
Window
Help
Debug
.NET
Default.aspx.cs*
Default.aspx
Toolbox
Server Explorer
_Default
Page_Load(object sender, EventArgs e)
Color
Doors
Equals
GetHashCode
GetType
Go
ToString
int EvjenCar.Go()
Solution Explorer - Solutio...
Solution 'ExampleBP' (1 project)
C:\...\ExampleBP\
App_Data
CarDetails.car
Bin
CarBuildProvider.dll
CarBuildProvider.pdb
Default.aspx
Default.aspx.cs
web.config
Solution Explorer
Class View
Properties
Ln 20
Col 1
Ch 1
INS

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Xml.Linq;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        EvjenCar newCar = new EvjenCar();
        Response.Write(newCar.Color + "<br />");
        Response.Write(newCar.Doors.ToString() + "<br />");
        Response.Write(newCar.
    }
}
```

Figure 1-15

In addition to getting access to the properties and methods of the class, you also gain access to the values that are defined in the `.car` file. This is shown in Figure 1-16. The simple code example shown in Figure 1-15 is used for this browser output.


Untitled Page - Windows Internet Explorer
http://localhost:50709/ExampleBP/Default.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

```
Blue
4
150
```

Figure 1-16

Although a car class is not the most useful thing in the world, this example shows you how to take the build provider mechanics into your own hands to extend your application's capabilities.

Global.asax

If you add a new item to your ASP.NET application, you get the Add New Item dialog. From here, you can see that you can add a Global Application Class to your applications. This adds a `Global.asax` file. This file is used by the application to hold application-level events, objects, and variables — all of which are accessible application-wide. Active Server Pages developers had something similar with the `Global.asa` file.

Your ASP.NET applications can have only a single `Global.asax` file. This file supports a number of items. When it is created, you are given the following template:

```
<%@ Application Language="VB" %>

<script runat="server">

    Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' Code that runs on application startup
    End Sub

    Sub Application_End(ByVal sender As Object, ByVal e As EventArgs)
        ' Code that runs on application shutdown
    End Sub

    Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
        ' Code that runs when an unhandled error occurs
    End Sub
    Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' Code that runs when a new session is started
    End Sub

    Sub Session_End(ByVal sender As Object, ByVal e As EventArgs)
        ' Code that runs when a session ends.
        ' Note: The Session_End event is raised only when the sessionstate mode
        ' is set to InProc in the Web.config file. If session mode is
        ' set to StateServer
        ' or SQLServer, the event is not raised.
    End Sub

</script>
```

Just as you can work with page-level events in your `.aspx` pages, you can work with overall application events from the `Global.asax` file. In addition to the events listed in this code example, the following list details some of the events you can structure inside this file:

- ❑ `Application_Start`: Called when the application receives its very first request. It is an ideal spot in your application to assign any application-level variables or state that must be maintained across all users.
- ❑ `Session_Start`: Similar to the `Application_Start` event except that this event is fired when an individual user accesses the application for the first time. For instance, the `Application_Start` event fires once when the first request comes in, which gets the application going, but the `Session_Start` is invoked for each end user who requests something from the application for the first time.
- ❑ `Application_BeginRequest`: Although it not listed in the preceding template provided by Visual Studio 2008, the `Application_BeginRequest` event is triggered before each and every request that comes its way. This means that when a request comes into the server, before this request is processed, the `Application_BeginRequest` is triggered and dealt with before any processing of the request occurs.
- ❑ `Application_AuthenticateRequest`: Triggered for each request and enables you to set up custom authentications for a request.
- ❑ `Application_Error`: Triggered when an error is thrown anywhere in the application by any user of the application. This is an ideal spot to provide application-wide error handling or an event recording the errors to the server's event logs.
- ❑ `Session_End`: When running in `InProc` mode, this event is triggered when an end user leaves the application.
- ❑ `Application_End`: Triggered when the application comes to an end. This is an event that most ASP.NET developers won't use that often because ASP.NET does such a good job of closing and cleaning up any objects that are left around.

In addition to the global application events that the `Global.asax` file provides access to, you can also use directives in this file as you can with other ASP.NET pages. The `Global.asax` file allows for the following directives:

- ❑ `@Application`
- ❑ `@Assembly`
- ❑ `@Import`

These directives perform in the same way when they are used with other ASP.NET page types.

An example of using the `Global.asax` file is shown in Listing 1-22. It demonstrates how to log when the ASP.NET application domain shuts down. When the ASP.NET application domain shuts down, the ASP.NET application abruptly comes to an end. Therefore, you should place any logging code in the `Application_End` method of the `Global.asax` file.

Listing 1-22: Using the Application_End event in the Global.asax file

VB

```
<%@ Application Language="VB" %>
<%@ Import Namespace="System.Reflection" %>
<%@ Import Namespace="System.Diagnostics" %>

<script runat="server">

    Sub Application_End(ByVal sender As Object, ByVal e As EventArgs)
        Dim MyRuntime As HttpRuntime = _
           GetType(System.Web.HttpRuntime).InvokeMember("_theRuntime", _
           BindingFlags.NonPublic Or BindingFlags.Static Or _
           BindingFlags.GetField, _
           Nothing, Nothing, Nothing)

        If (MyRuntime Is Nothing) Then
            Return
        End If

        Dim shutDownMessage As String = _
           CType(MyRuntime.GetType().InvokeMember("_shutDownMessage", _
           BindingFlags.NonPublic Or BindingFlags.Instance Or
           BindingFlags.GetField, _
           Nothing, MyRuntime, Nothing), System.String)

        Dim shutDownStack As String = _
           CType(MyRuntime.GetType().InvokeMember("_shutDownStack", _
           BindingFlags.NonPublic Or BindingFlags.Instance Or
           BindingFlags.GetField, _
           Nothing, MyRuntime, Nothing), System.String)

        If (Not EventLog.SourceExists(".NET Runtime")) Then
            EventLog.CreateEventSource(".NET Runtime", "Application")
        End If

        Dim logEntry As EventLog = New EventLog()
        logEntry.Source = ".NET Runtime"
        logEntry.WriteEntry(String.Format(_
           "shutDownMessage={0}\r\n\r\n_shutDownStack={1}", _
           shutDownMessage, shutDownStack), EventLogEntryType.Error)
    End Sub

</script>
```

C#

```
<%@ Application Language="C#" %>
```

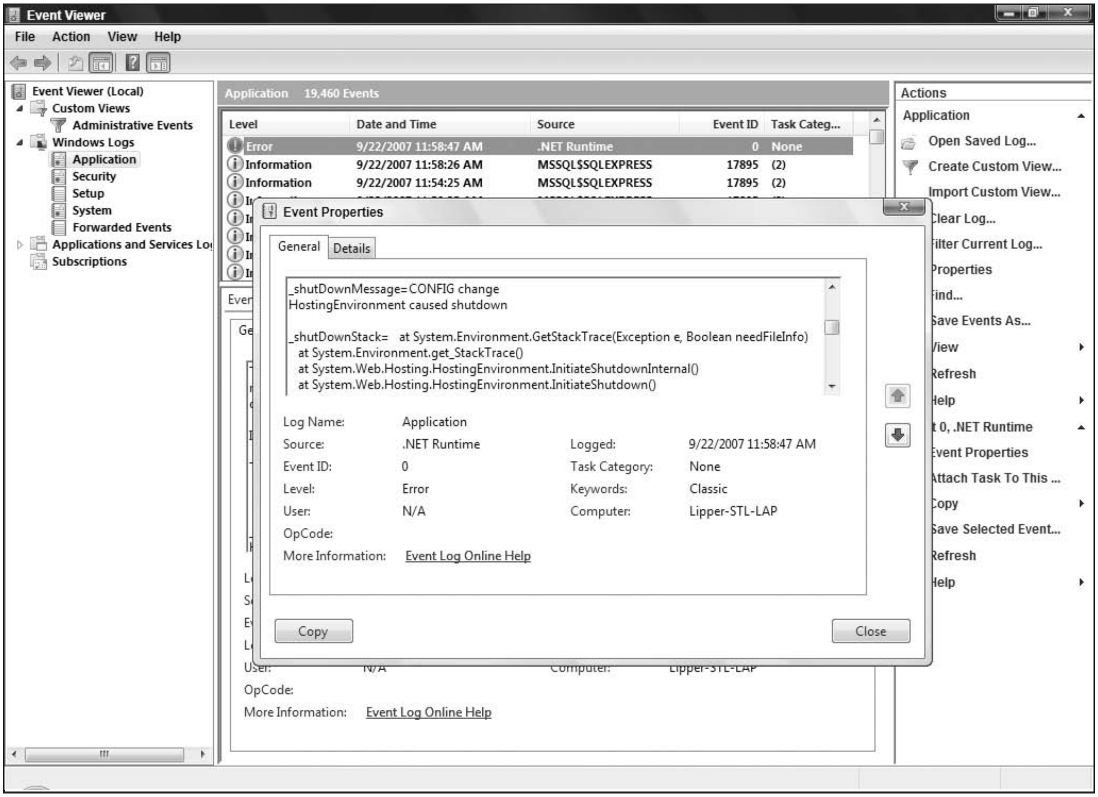
Continued

```
<%@ Import Namespace="System.Reflection" %>
<%@ Import Namespace="System.Diagnostics" %>

<script runat="server">

    void Application_End(object sender, EventArgs e)
    {
        HttpRuntime runtime =
           (HttpRuntime)typeof(System.Web.HttpRuntime).InvokeMember("_theRuntime",
           BindingFlags.NonPublic | BindingFlags.Static | BindingFlags.GetField,
           null, null, null);

        if (runtime == null)
        {
            return;
        }

        string shutDownMessage =
           (string)runtime.GetType().InvokeMember("_shutDownMessage",
           BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.GetField,
           null, runtime, null);

        string shutDownStack =
           (string)runtime.GetType().InvokeMember("_shutDownStack",
           BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.GetField,
           null, runtime, null);

        if (!EventLog.SourceExists(".NET Runtime"))
        {
            EventLog.CreateEventSource(".NET Runtime", "Application");
        }

        EventLog logEntry = new EventLog();
        logEntry.Source = ".NET Runtime";
        logEntry.WriteEntry(String.Format("\r\n\r\n_" +
           "shutDownMessage={0}\r\n\r\n_shutDownStack={1}",
           shutDownMessage, shutDownStack), EventLogEntryType.Error);
    }

</script>
```

With this code in place in your `Global.asax` file, start your ASP.NET application. Next, do something to cause the application to restart. You could, for example, make a change to the `web.config` file while the application is running. This triggers the `Application_End` event, and you see the following addition (shown in Figure 1-17) to the event log.

Working with Classes Through VS2008

This chapter showed you how to work with classes within your ASP.NET projects. In constructing and working with classes, you will find that Visual Studio 2008 is quite helpful. Two particularly useful items are a class designer file and an Object Test Bench. The class designer file has an extension of `.cd` and


Event Viewer
File
Action
View
Help
Event Viewer (Local)
Custom Views
Administrative Events
Windows Logs
Application
Security
Setup
System
Forwarded Events
Applications and Services Lo
Subscriptions
Application 19,460 Events
Actions
Application
Open Saved Log...
Create Custom View...
Import Custom View...
Clear Log...
Filter Current Log...
Properties
Find...
Save Events As...
View
Refresh
Help
Event 0, .NET Runtime
Event Properties
Attach Task To This ...
Copy
Save Selected Event...
Refresh
Help
Event Properties
General
Details
Copy
Close

Level	Date and Time	Source	Event ID	Task Categ...
Error	9/22/2007 11:58:47 AM	.NET Runtime	0	None
Information	9/22/2007 11:58:26 AM	MSSQL$SQLEXPRESS	17895	(2)
Information	9/22/2007 11:54:25 AM	MSSQL$SQLEXPRESS	17895	(2)

```
_shutDownMessage=CONFIG change
HostingEnvironment caused shutdown

_shutDownStack=   at System.Environment.GetStackTrace(Exception e, Boolean needFileInfo)
   at System.Environment.get_StackTrace()
   at System.Web.Hosting.HostingEnvironment.InitiateShutdownInternal()
   at System.Web.Hosting.HostingEnvironment.InitiateShutdown()
```

Log Name: Application
Source: .NET Runtime
Logged: 9/22/2007 11:58:47 AM
Event ID: 0
Task Category: None
Level: Error
Keywords: Classic
User: N/A
Computer: Lipper-STL-LAP
OpCode:
More Information: Event Log Online Help

Figure 1-17

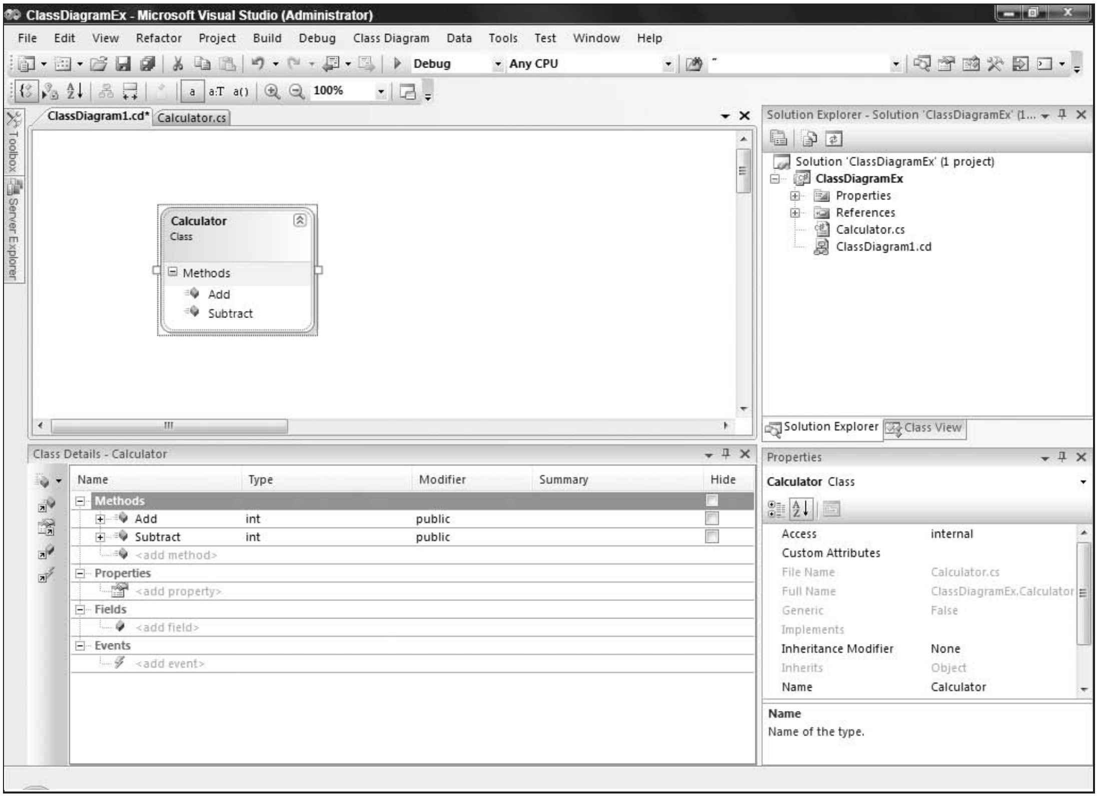
gives you a visual way to view your class, as well as all the available methods, properties, and other class items it contains. The Object Test Bench built into Visual Studio gives you a way to instantiate your classes and test them without creating a test application, a task which can be quite time consuming.

To see these items in action, create a new Class Library project in the language of your choice. This project has a single class file, `Class1.vb` or `.cs`. Delete this file and create a new class file called `Calculator.vb` or `.cs`, depending on the language you are using. From here, complete the class by creating a simple `Add()` and `Subtract()` method. Each of these methods takes in two parameters (of type `Integer`) and returns a single Integer with the appropriate calculation performed.

After you have the `Calculator` class in place, the easiest way to create your class designer file for this particular class is to right-click on the `Calculator.vb` file directly in the Solution Explorer and select View Class Diagram from the menu. This creates a `ClassDiagram1.cd` file in your solution.

The visual file, `ClassDiagram1.cd`, is presented in Figure 1-18.

The new class designer file gives you a design view of your class. In the Document Window of Visual Studio, you see a visual representation of the `Calculator` class. The class is represented in a box and


ClassDiagramEx - Microsoft Visual Studio (Administrator)
File Edit View Refactor Project Build Debug Class Diagram Data Tools Test Window Help
Debug
Any CPU
100%
ClassDiagram1.cd*
Calculator.cs
Toolbox
Server Explorer
Calculator
Class
Methods
Add
Subtract
Solution Explorer - Solution 'ClassDiagramEx' (1...
Solution 'ClassDiagramEx' (1 project)
ClassDiagramEx
Properties
References
Calculator.cs
ClassDiagram1.cd
Solution Explorer
Class View
Class Details - Calculator
Properties
Calculator Class

Name	Type	Modifier	Summary	Hide
Methods				☐
Add	int	public		☐
Subtract	int	public		☐
<add method>				
Properties				
<add property>				
Fields				
<add field>				
Events				
<add event>				

Access	internal
Custom Attributes	
File Name	Calculator.cs
Full Name	ClassDiagramEx.Calculator
Generic	False
Implements	
Inheritance Modifier	None
Inherits	Object
Name	Calculator

Name
Name of the type.

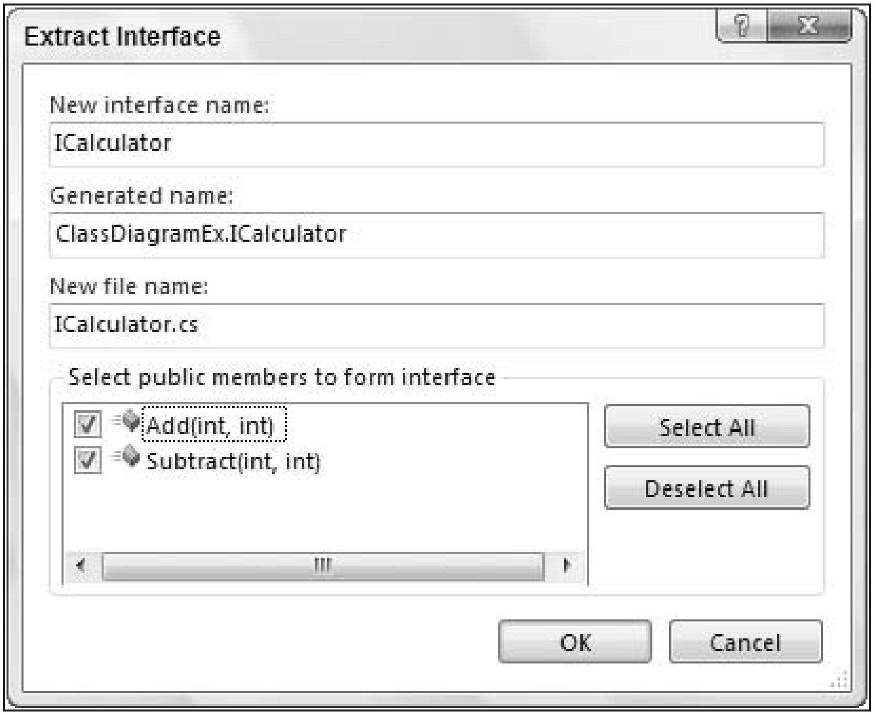
Figure 1-18

provides the name of the class, as well as two available methods that are exposed by the class. Because of the simplicity of this class, the details provided in the visual view are limited.

You can add additional classes to this diagram simply by dragging and dropping class files onto the design surface. You can then arrange the class files on the design surface as you wish. A connection is in place for classes that are inherited from other class files or classes that derive from an interface or abstract class. In fact, you can extract an interface from the class you just created directly in the class designer by right-clicking on the Calculator class box and selecting Refactor ➪ Extract Interface from the provided menu. This launches the Extract Interface dialog that enables you to customize the interface creation. This dialog box is presented in Figure 1-19.

After you click OK, the `ICalculator` interface is created and is then visually represented in the class diagram file, as illustrated in Figure 1-20.

In addition to creating items such as interfaces on-the-fly, you can also modify your `Calculator` class by adding additional methods, properties, events, and more through the Class Details pane found in Visual Studio. The Class Details pane is presented in Figure 1-21.



Extract Interface

New interface name: ICalculator

Generated name: ClassDiagramEx.ICalculator

New file name: ICalculator.cs

Select public members to form interface

☑ Add(int, int)
☑ Subtract(int, int)

Figure 1-19

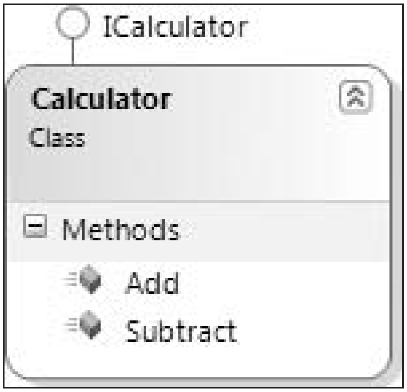
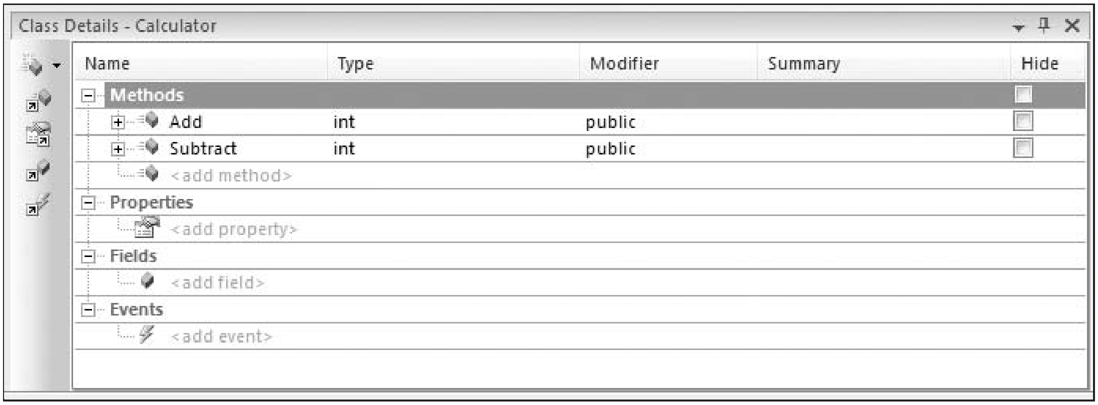


Figure 1-20

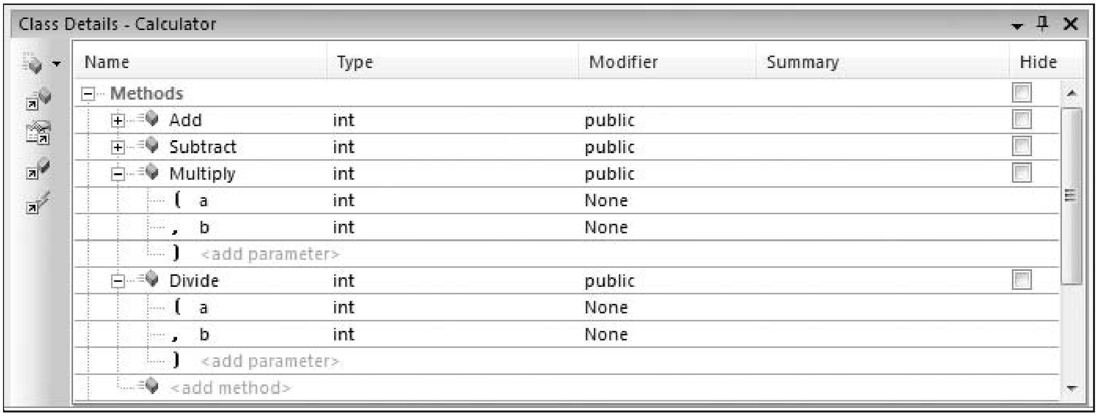
From this view of the class, you can directly add any additional methods, properties, fields, or events without directly typing code in your class file. When you enter these items in the Class Details view, Visual Studio generates the code for you on your behalf. For an example of this, add the additional `Multiply()` and `Divide()` methods that the `Calculator` class needs. Expanding the plus sign next to these methods shows the parameters needed in the signature. This is where you add the required `a` and `b` parameters. When you have finished, your Class Details screen should appear as shown in Figure 1-22.



Class Details - Calculator

Name	Type	Modifier	Summary	Hide
Methods				☐
Add	int	public		☐
Subtract	int	public		☐
<add method>				
Properties				
<add property>				
Fields				
<add field>				
Events				
<add event>				

Figure 1-21



Class Details - Calculator

Name	Type	Modifier	Summary	Hide
Methods				☐
Add	int	public		☐
Subtract	int	public		☐
Multiply	int	public		☐
(a	int	None		
, b	int	None		
) <add parameter>				
Divide	int	public		☐
(a	int	None		
, b	int	None		
) <add parameter>				
<add method>				

Figure 1-22

After you have added new `Multiply()` and `Divide()` methods and the required parameters, you see that the code in the `Calculator` class has changed to indicate these new methods are present. When the framework of the method is in place, you also see that the class has not been implemented in any fashion. The C# version of the `Multply()` and `Divide()` methods created by Visual Studio is presented in Listing 1-23.

Listing 1-23: The framework provided by Visual Studio's class designer

```
public int Multiply(int a, int b)
{
   throw new System.NotImplementedException();
}

public int Divide(int a, int b)
{
   throw new System.NotImplementedException();
}
```

The new class designer files give you a powerful way to view and understand your classes better — sometimes a picture really is worth a thousand words. One interesting last point on the `.cd` file is that Visual Studio is really doing all the work with this file. If you open the `ClassDesigner1.cd` file in Notepad, you see the results presented in Listing 1-24.

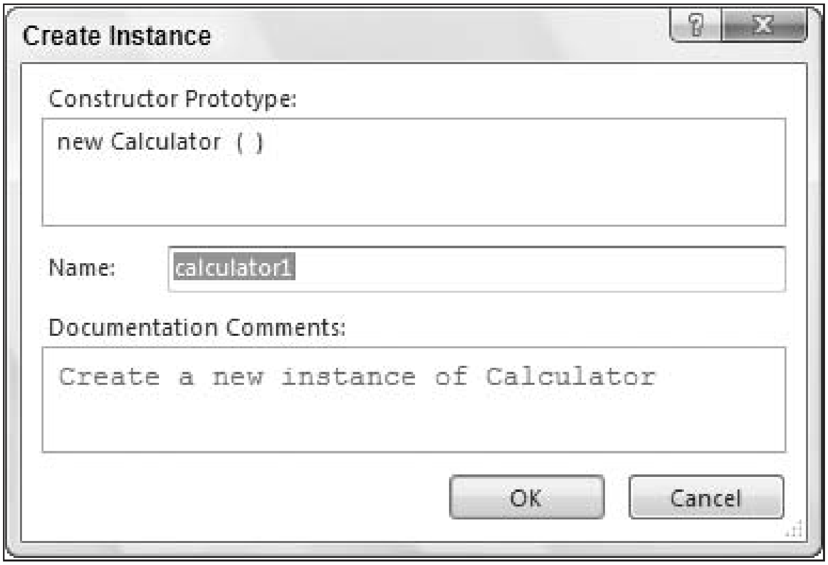
Listing 1-24: The real ClassDesigner1.cd file as seen in Notepad

```
<?xml version="1.0" encoding="utf-8"?>
<ClassDiagram MajorVersion="1" MinorVersion="1">
  <Class Name="ClassDiagramEx.Calculator">
    <Position X="1.25" Y="0.75" Width="1.5" />
    <TypeIdentifier>
      <HashCode>AAIAAAAAAQAAAAAAAAADAAAAAAAAAAAAAAAAAAAAAA=</HashCode>
      <FileName>Calculator.cs</FileName>
    </TypeIdentifier>
    <Lollipop Position="0.2" />
  </Class>
  <Font Name="Segoe UI" Size="8.25" />
</ClassDiagram>
```

As you can see, it is a rather simple XML file that defines the locations of the class and the items connected to the class.

In addition to using the new class designer to provide a visual representation of your classes, you can also use it to instantiate and test your new objects. To do this, right-click on the `Calculator` class file in the `ClassDiagram1.cd` file and select Create Instance➪Calculator() from the provided menu.

This launches the Create Instance dialog that simply asks you to create a new name for your class instantiation. This dialog is illustrated in Figure 1-23.



new Calculator ( )

Name: calculator1

Documentation Comments:

Create a new instance of Calculator

Figure 1-23

From here, click OK and you see a visual representation of this instantiation in the new Object Test Bench directly in Visual Studio. The Object Test Bench now contains only a single gray box classed calculator1. Right-click on this object directly in the Object Test Bench, and select Invoke Method➪Add(int, int) from the provided menu. This is illustrated in Figure 1-24.

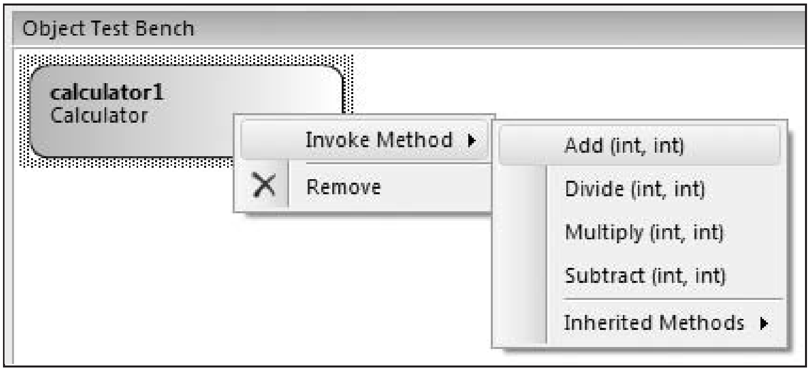
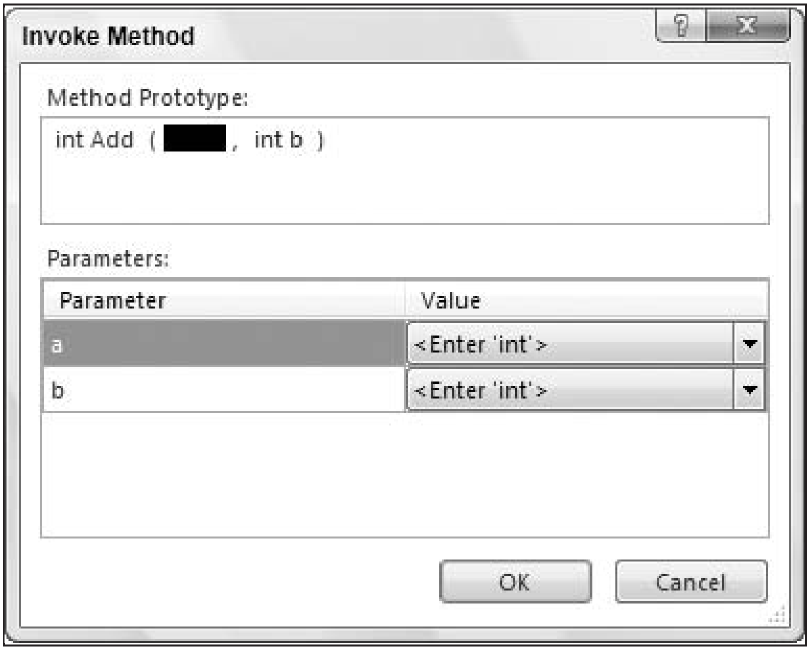

Object Test Bench
calculator1
Calculator
Invoke Method
Remove
Add (int, int)
Divide (int, int)
Multiply (int, int)
Subtract (int, int)
Inherited Methods

Figure 1-24

Selecting the Add method launches another dialog — the Invoke Method dialog. This dialog enables you to enter values for the required parameters, as shown in Figure 1-25.

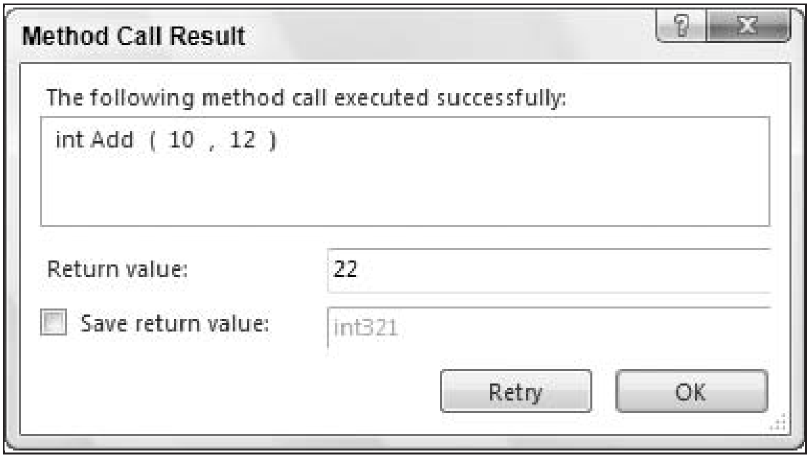

Invoke Method
Method Prototype:
Parameters:
OK
Cancel

int Add ([redacted] , int b)

Parameter	Value
a	<Enter 'int'>
b	<Enter 'int'>

Figure 1-25

After providing values and clicking OK, you see another dialog that provides you with the calculated result, as shown in Figure 1-26.



The following method call executed successfully:

```
int Add ( 10 , 12 )
```

Return value: 22

☐ Save return value: int321

Figure 1-26

This is a simple example. When you start working with more complex objects and collections, however, this feature is even more amazing because the designer enables you to work through the entire returned result visually directly in the IDE.

Summary

This chapter covered a lot of ground. It discussed some of the issues concerning ASP.NET applications as a whole and the choices you have when building and deploying these new applications. With the help of Visual Studio 2008, you now have options about which Web server to use when building your application and whether to work locally or remotely through the new built-in FTP capabilities.

ASP.NET 3.5 and Visual Studio 2008 make it easy to build your pages using an inline coding model or to select a new and better code-behind model that is simpler to use and easier to deploy. You also learned about the new cross-posting capabilities and the new fixed folders that ASP.NET 3.5 has incorporated to make your life easier. These folders make their resources available dynamically with no work on your part. You saw some of the outstanding new compilation options that you have at your disposal. Finally, you looked at ways in which Visual Studio 2008 makes it easy to work with the classes of your project.

As you worked through some of the examples, you may have been thinking, ‘‘WOW!’’ But wait . . . there’s plenty more to come!

2

ASP.NET Server Controls and Client-Side Scripts

As discussed in the previous chapter, ASP.NET evolved from Microsoft's earlier Web technology called Active Server Pages (referred to as *ASP* then and *classic ASP* today). This model was completely different from today's ASP.NET. Classic ASP used interpreted languages to accomplish the construction of the final HTML document before it was sent to the browser. ASP.NET, on the other hand, uses true compiled languages to accomplish the same task. The idea of building Web pages based on objects in a compiled environment is one of the main focuses of this chapter.

This chapter looks at how to use a particular type of object in ASP.NET pages called a *server control*, and how you can profit from using this control. We also introduce a particular type of server control — the HTML server control. The chapter also demonstrates how you can use JavaScript in ASP.NET pages to modify the behavior of server controls.

The rest of this chapter shows you how to use and manipulate server controls, both visually and programmatically, to help with the creation of your ASP.NET pages.

ASP.NET Server Controls

In the past, one of the difficulties of working with classic ASP was that you were completely in charge of the entire HTML output from the browser by virtue of the server-side code you wrote. Although this might seem ideal, it created a problem because each browser interpreted the HTML given to it in a slightly different manner.

The two main browsers out there at the time were Microsoft's Internet Explorer and Netscape Navigator. This meant that not only did developers have to be cognizant of the browser type to which they were outputting HTML, but they also had to take into account which versions of those particular browsers might be making a request to their application. Some developers resolved the issue by creating two separate applications. When an end user made an initial request to the application, the code made a browser check to see what browser type was making the request. Then, the ASP page would redirect the request down one path for an IE user, or down another path for a Netscape user.

Because requests came from so many different versions of the same browser, the developer often designed for the lowest possible version that might be used to visit the site. Essentially, everyone lost out by using the lowest common denominator as the target. This technique ensured that the page was rendered properly in most browsers making a request, but it also forced the developer to dumb-down his application. If applications were always built for the lowest common denominator, the developer could never take advantage of some of the more advanced features offered by newer browser versions.

ASP.NET server controls overcome these obstacles. When using the server controls provided by ASP.NET, you are not specifying the HTML to be output from your server-side code. Rather, you are specifying the functionality you want to see in the browser and letting the ASP.NET decide for you on the output to be sent to the browser.

When a request comes in, ASP.NET examines the request to see which browser type is making the request, as well as the version of the browser, and then it produces HTML output specific to that browser. This process is accomplished by processing a User Agent header retrieved from the HTTP Request to *sniff* the browser. This means that you can now build for the best browsers out there without worrying about whether features will work in the browsers making requests to your applications. Because of the previously described capabilities, you will often hear these controls referred to as *smart controls*.

Types of Server Controls

ASP.NET provides two distinct types of server controls — HTML server controls and Web server controls. Each type of control is quite different and, as you work with ASP.NET, you will see that much of the focus is on the Web server controls. This does not mean that HTML server controls have no value. They do provide you with many capabilities — some that Web server controls do not give you.

You might be asking yourself which is the better control type to use. The answer is that it really depends on what you are trying to achieve. HTML server controls map to specific HTML elements. You can place an `HtmlTable` server control on your ASP.NET page that works dynamically with a `<table>` element. On the other hand, Web server controls map to specific functionality that you want on your ASP.NET pages. This means an `<asp:Panel>` control might use a `<table>` or an another element altogether — it really depends on the capability of the browser making the request.

The following table provides a summary of information on when to use HTML server controls and when to use Web server controls.

Control Type	When to Use This Control Type
HTML Server	When converting traditional ASP 3.0 Web pages to ASP.NET Web pages and speed of completion is a concern. It is a lot easier to change your HTML elements to HTML server controls than it is to change them to Web server controls. When you prefer a more HTML-type programming model. When you want to explicitly control the code that is generated for the browser.
Web Server	When you require a richer set of functionality to perform complicated page requirements. When you are developing Web pages that will be viewed by a multitude of browser types and that require different code based upon these types. When you prefer a more Visual Basic–type programming model that is based on the use of controls and control properties.

Of course, some developers like to separate certain controls from the rest and place them in their own categories. For instance, you may see references to the following types of controls:

- ☐ **List controls:** These control types allow data to be bound to them for display purposes of some kind.
- ☐ **Rich controls:** Controls, such as the Calendar control, that display richer content and capabilities than other controls.
- ☐ **Validation controls:** Controls that interact with other form controls to validate the data that they contain.
- ☐ **Mobile controls:** Controls that are specific for output to devices such as mobile phones, PDAs, and more.
- ☐ **User controls:** These are not really controls, but page templates that you can work with as you would a control on your ASP.NET page.
- ☐ **Custom controls:** Controls that you build yourself and use in the same manner as the supplied ASP.NET server controls that come with the default install of ASP.NET 3.5.

When you are deciding between HTML server controls and Web server controls, remember that no hard and fast rules exist about which type to use. You might find yourself working with one control type more than another, but certain features are available in one control type that might not be available in the other. If you are trying to accomplish a specific task and you do not see a solution with the control type you are using, take a look at the other control type because it may very well hold the answer. Also, realize that you can mix and match these control types. Nothing says that you cannot use both HTML server controls and Web server controls on the same page or within the same application.

Building with Server Controls

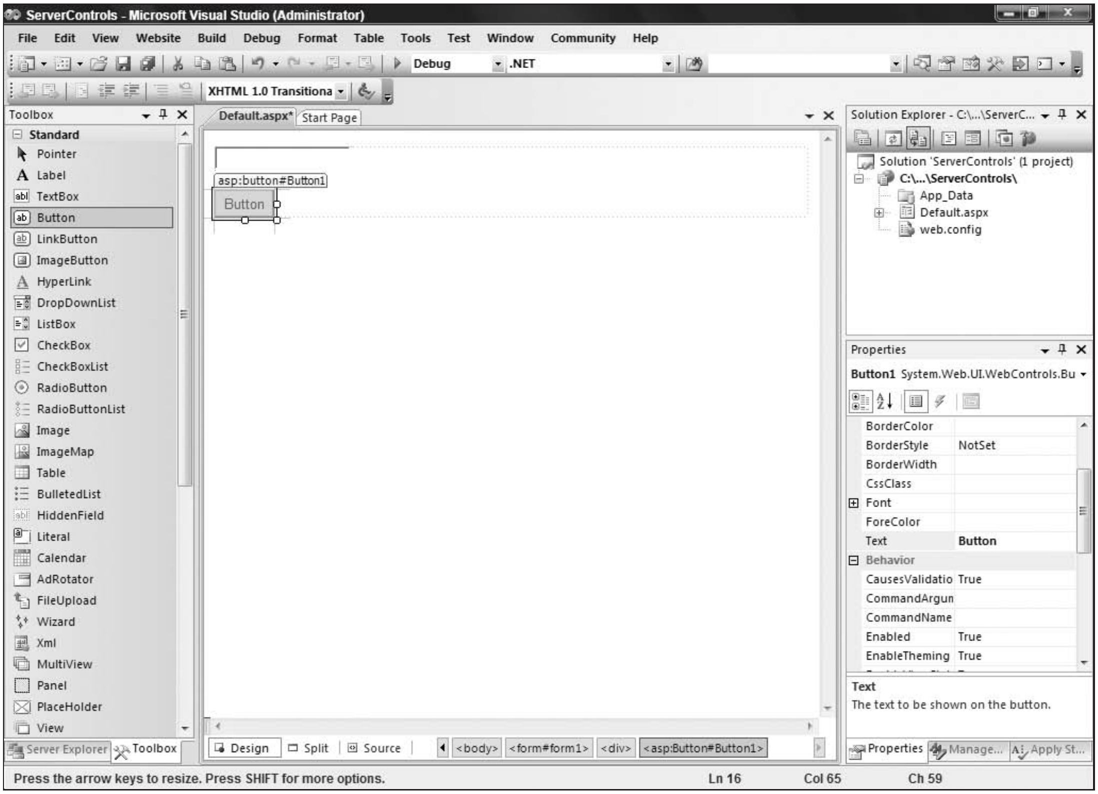
You have a couple of ways to use server controls to construct your ASP.NET pages. You can actually use tools that are specifically designed to work with ASP.NET 3.5 that enable you to visually drag and drop controls onto a design surface and manipulate the behavior of the control. You can also work with server controls directly through code input.

Working with Server Controls on a Design Surface

Visual Studio 2008 enables you to visually create an ASP.NET page by dragging and dropping visual controls onto a design surface. You can get to this visual design option by clicking the Design tab at the bottom of the IDE when viewing your ASP.NET page. You can also show the Design view and the Source code view in the same document window. This is a new feature available in Visual Studio 2008. When the Design view is present, you can place the cursor on the page in the location where you want the control to appear and then double-click the control you want in the Toolbox window of Visual Studio. Unlike the 2002 and 2003 versions of Visual Studio, Visual Studio 2008 does a really good job (as does the previous Visual Studio 2005) of not touching your code when switching between the Design and Source tabs.

In the Design view of your page, you can highlight a control and the properties for the control appear in the Properties window. For example, Figure 2-1 shows a Button control selected in the design panel, and its properties are displayed in the Properties window on the lower right.

Changing the properties in the window changes the appearance or behavior of the highlighted control. Because all controls inherit from a specific base class (`WebControl`), you can also highlight multiple controls at the same time and change the base properties of all the controls at once. You do this by holding down the Ctrl key as you make your control selections.


ServerControls - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Format Table Tools Test Window Community Help
Debug
.NET
XHTML 1.0 Transitiona
Toolbox
Standard
Pointer
Label
TextBox
Button
LinkButton
ImageButton
HyperLink
DropDownList
ListBox
CheckBox
CheckBoxList
RadioButton
RadioButtonList
Image
ImageMap
Table
BulletedList
HiddenField
Literal
Calendar
AdRotator
FileUpload
Wizard
Xml
MultiView
Panel
PlaceHolder
View
Server Explorer
Toolbox
Default.aspx*
Start Page
asp:button#Button1
Button
Design
Split
Source
<body>
<form#form1>
<div>
<asp:Button#Button1>
Solution Explorer - C:\...\ServerC...
Solution 'ServerControls' (1 project)
C:\...\ServerControls\
App_Data
Default.aspx
web.config
Properties
Button1 System.Web.UI.WebControls.Bu
Properties
Manage...
Apply St...
Ln 16
Col 65
Ch 59

BorderColor	
BorderStyle	NotSet
BorderWidth	
CssClass	
⊞ Font	
ForeColor	
Text	Button
⊟ Behavior	
CausesValidatio	True
CommandArgun	
CommandName	
Enabled	True
EnableTheming	True

Text
The text to be shown on the button.

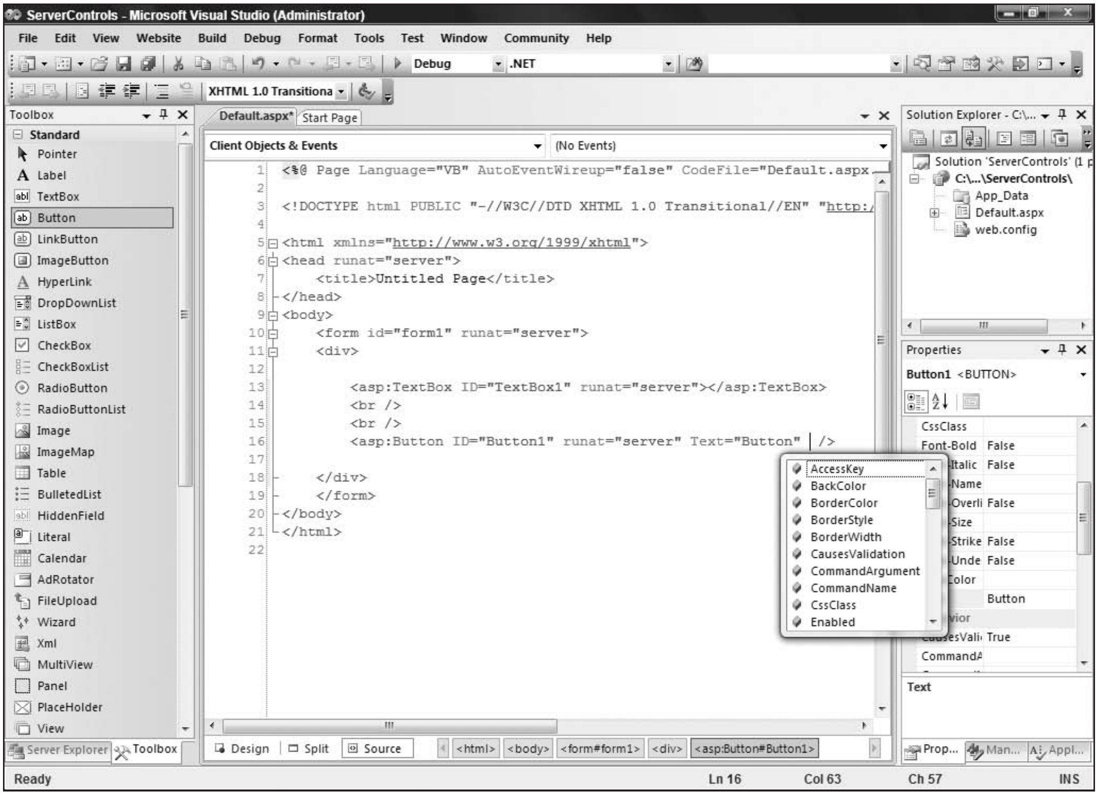
Press the arrow keys to resize. Press SHIFT for more options.

Figure 2-1

Coding Server Controls

You also can work from the code page directly. Because many developers prefer this, it is the default when you first create your ASP.NET page. Hand-coding your own ASP.NET pages may seem to be a slower approach than simply dragging and dropping controls onto a design surface, but it isn't as slow as you might think. You get plenty of assistance in coding your applications from Visual Studio 2008. As you start typing in Visual Studio, the IntelliSense features kick in and help you with code auto-completion. Figure 2-2, for example, shows an IntelliSense drop-down list of possible code completion statements that appeared as the code was typed.

The IntelliSense focus is on the most commonly used attribute or statement for the control or piece of code that you are working with. Using IntelliSense effectively as you work is a great way to code with great speed.



ServerControls - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Format Tools Test Window Community Help
Debug
.NET
XHTML 1.0 Transitiona
Toolbox
Standard
Pointer
Label
TextBox
Button
LinkButton
ImageButton
HyperLink
DropDownList
ListBox
CheckBox
CheckBoxList
RadioButton
RadioButtonList
Image
ImageMap
Table
BulletedList
HiddenField
Literal
Calendar
AdRotator
FileUpload
Wizard
Xml
MultiView
Panel
PlaceHolder
View
Server Explorer
Toolbox
Default.aspx*
Start Page
Client Objects & Events
(No Events)
AccessKey
BackColor
BorderColor
BorderStyle
BorderWidth
CausesValidation
CommandArgument
CommandName
CssClass
Enabled
Design
Split
Source
<html>
<body>
<form#form1>
<div>
<asp:Button#Button1>
Solution Explorer - C:\...
Solution 'ServerControls' (1 p
C:\...\ServerControls\
App_Data
Default.aspx
web.config
Properties
Button1 <BUTTON>
CssClass
Font-Bold False
Italic False
Name
Overli False
Size
Strike False
Unde False
olor
Button
vior
sVali True
CommandA
Text
Prop...
Man...
Appl...
Ready
Ln 16
Col 63
Ch 57
INS

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http:/

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <br />
        <br />
        <asp:Button ID="Button1" runat="server" Text="Button" />

    </div>
    </form>
</body>
</html>
```

Figure 2-2

As with Design view, the Source view of your page lets you drag and drop controls from the Toolbox onto the code page itself. For example, dragging and dropping a TextBox control onto the code page produces the same results as dropping it on the design page:

```
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
```

You can also highlight a control in Source view or simply place your cursor in the code statement of the control, and the Properties window displays the properties of the control. Now, you can apply properties directly in the Properties window of Visual Studio, and these properties are dynamically added to the code of your control.

Working with Server Control Events

As discussed in Chapter 1, ASP.NET uses more of a traditional Visual Basic event model than classic ASP. Instead of working with interpreted code, you are actually coding an event-based structure for your pages. Classic ASP used an interpreted model — when the server processed the Web page, the code of the page was interpreted line-by-line in a linear fashion where the only ''event'' implied was the page loading. This meant that occurrences you wanted to get initiated early in the process were placed at the top of the page.

Today, ASP.NET uses an event-driven model. Items or coding tasks get initiated only when a particular event occurs. A common event in the ASP.NET programming model is `Page_Load`, which is illustrated in Listing 2-1.

Listing 2-1: Working with specific page events

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   ' Code actions here
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
   // Code actions here
}
```

Not only can you work with the overall page — as well as its properties and methods at particular moments in time through page events — but you can also work with the server controls contained on the page through particular control events. For example, one common event for a button on a form is `Button_Click`, which is illustrated in Listing 2-2.

Listing 2-2: Working with a Button Click event

VB

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
   ' Code actions here
End Sub
```

C#

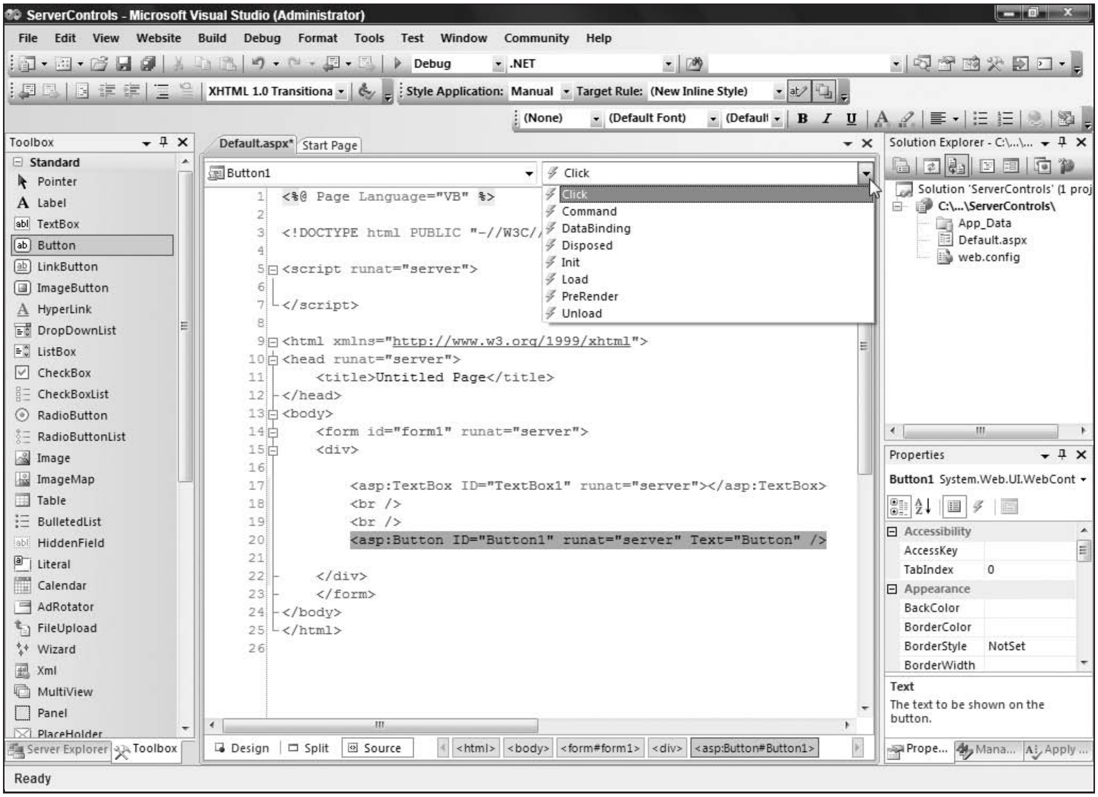
```
protected void Button1_Click(object sender, EventArgs e)
{
   // Code actions here
}
```

The event shown in Listing 2-2 is fired only when the end user actually clicks the button on the form that has an `OnClick` attribute value of `Button1_Click`. Therefore, not only does the event handler exist in the server-side code of the ASP.NET page, but that handler is also hooked up using the `OnClick` property of the server control in the associated ASP.NET page markup, as illustrated in the following code:

```
<asp:Button ID="Button1" Runat="server" Text="Button" OnClick="Button1_Click" />
```

How do you fire these events for server controls? You have a couple of ways to go about it. The first way is to pull up your ASP.NET page in the Design view and double-click the control for which you want to create a server-side event. For instance, double-clicking a Button server control in Design view creates the structure of the `Button1_Click` event within your server-side code, whether the code is in a code-behind file or inline. This creates a stub handler for that server control's most popular event.

With that said, be aware that a considerable number of additional events are available to the Button control that you cannot get at by double-clicking the control. To access them, pull up the page that contains the server-side code, select the control from the first drop-down list at the top of the IDE, and then choose the particular event you want for that control in the second drop-down list. Figure 2-3 shows the event drop-down list displayed. You might, for example, want to work with the Button control's `PreRender` event rather than its `Click` event. The handler for the event you choose is placed in your server-side code.


ServerControls - Microsoft Visual Studio (Administrator)
File
Edit
View
Website
Build
Debug
Format
Tools
Test
Window
Community
Help
Debug
.NET
XHTML 1.0 Transitiona
Style Application: Manual
Target Rule: (New Inline Style)
(None)
(Default Font)
(Default
Toolbox
Standard
Pointer
Label
TextBox
Button
LinkButton
ImageButton
HyperLink
DropDownList
ListBox
CheckBox
CheckBoxList
RadioButton
RadioButtonList
Image
ImageMap
Table
BulletedList
HiddenField
Literal
Calendar
AdRotator
FileUpload
Wizard
Xml
MultiView
Panel
PlaceHolder
Server Explorer
Toolbox
Default.aspx*
Start Page
Button1
Click
Click
Command
DataBinding
Disposed
Init
Load
PreRender
Unload
Design
Split
Source
<html>
<body>
<form#form1>
<div>
<asp:Button#Button1>
Solution Explorer - C:\...\...
Solution 'ServerControls' (1 proj
C:\...\ServerControls\
App_Data
Default.aspx
web.config
Properties
Button1 System.Web.UI.WebCont
Accessibility
Appearance
Prope...
Mana...
Apply ...
Ready

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <br />
        <br />
        <asp:Button ID="Button1" runat="server" Text="Button" />

    </div>
    </form>
</body>
</html>
```

AccessKey	
TabIndex	0
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	

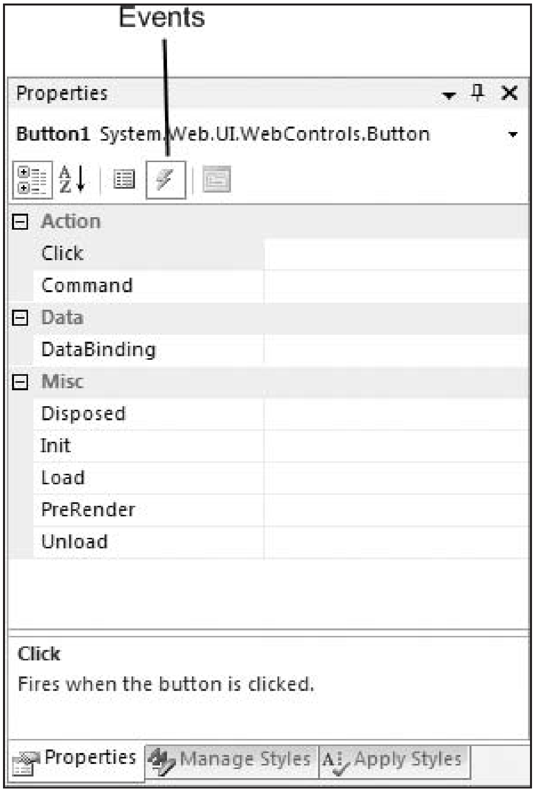
Text
The text to be shown on the button.

Figure 2-3

The second way is to bind to server-side events for your server controls from the Properties window of Visual Studio. This works only from Design view of the page. In Design view, highlight the server control that you want to work with. The properties for the control then appear in the Properties window, along with an icon menu. One of the icons, the Events icon, is represented by a lightning bolt (shown in Figure 2-4).

Clicking the Events icon pulls up a list of events available for the control. You simply double-click one of the events to get that event structure created in your server-side code.

After you have an event structure in place, you can program specific actions that you want to occur when the event is fired.


Events
Properties
Button1 System.Web.UI.WebControls.Button
Action
Click
Command
Data
DataBinding
Misc
Disposed
Init
Load
PreRender
Unload
Properties
Manage Styles
Apply Styles

Click
Fires when the button is clicked.

Figure 2-4

Applying Styles to Server Controls

More often than not, you want to change the default style (which is basically no style) to the server controls you implement in your applications. You most likely want to build your Web applications so that they reflect your own look-and-feel. One way to customize the appearance of the controls in your pages is to change the controls' properties.

As stated earlier in this chapter, to get at the properties of a particular control you simply highlight the control in the Design view of the page from Visual Studio. If you are working from the Source view, place the cursor in the code of the control. The properties presented in the Properties window allow you to control the appearance and behavior of the selected control.

Examining the Controls' Common Properties

Many of the default server controls that come with ASP.NET 3.5 are derived from the `WebControl` class and share similar properties that enable you to alter their appearance and behavior. Not all the derived controls use all the available properties (although many are implemented). Another important point is that not all server controls are implemented from the `WebControl` class. For instance, the Literal, PlaceHolder, Repeater, and XML server controls do not derive from the `WebControl` base class, but instead the `Control` class.

HTML server controls also do not derive from the `WebControl` base class because they are more focused on the set attributes of particular HTML elements. The following table lists the common properties the server controls share.

Property	**Description**
`AccessKey`	Enables you to assign a character to be associated with the Alt key so that the end user can activate the control using quick-keys on the keyboard. For instance, you can assign a Button control an `AccessKey` property value of `K`. Now, instead of clicking the button on the ASP.NET page (using a pointer controlled by the mouse), the end user can simply press Alt + K.
`Attributes`	Enables you to define additional attributes for a Web server control that are not defined by a public property.
`BackColor`	Controls the color shown behind the control's layout on the ASP.NET page.
`BorderColor`	Assigns a color that is shown around the physical edge of the server control.
`BorderWidth`	Assigns a value to the width of the line that makes up the border of the control. Placing a number as the value assigns the number as a pixel-width of the border. The default border color is black if the `BorderColor` property is not used in conjunction with the `BorderWidth` property setting.
`BorderStyle`	Enables you to assign the design of the border that is placed around the server control. By default, the border is created as a straight line, but a number of different styles can be used for your borders. Other possible values for the `BorderStyle` property include `Dotted`, `Dashed`, `Solid`, `Double`, `Groove`, `Ridge`, `Inset`, and `Outset`.
`CssClass`	Assigns a custom CSS (Cascading Style Sheet) class to the control.
`Enabled`	Enables you to turn off the functionality of the control by setting the value of this property to `False`. By default, the `Enabled` property is set to `True`.
`EnableTheming`	Enables you to turn on theming capabilities for the selected server control. The default value is `True`.
`Font`	Sets the font for all the text that appears anywhere in the control.
`ForeColor`	Sets the color of all the text that appears anywhere in the control.
`Height`	Sets the height of the control.
`SkinID`	Sets the skin to use when theming the control.
`Style`	Enables you to apply CSS styles to the control.
`TabIndex`	Sets the control's tab position in the ASP.NET page. This property works in conjunction with other controls on the page.
`ToolTip`	Assigns text that appears in a yellow box in the browser when a mouse pointer is held over the control for a short length of time. This can be used to add more instructions for the end user.
`Width`	Sets the width of the control.

You can see these common properties in many of the server controls you work with. Some of the properties of the `WebControl` class presented here work directly with the themeing system built into ASP.NET such as the `EnableTheming` and `SkinID` properties. These properties are covered in more detail in Chapter 6. You also see additional properties that are specific to the control you are viewing. Learning about the properties from the preceding table enables you to quickly work with Web server controls and to modify them to your needs.

Next take a look at some additional methods of customizing the look-and-feel of your server controls.

Changing Styles Using Cascading Style Sheets

One method of changing the look-and-feel of specific elements on your ASP.NET page is to apply a *style* to the element. The most rudimentary method of applying a defined look-and-feel to your page elements is to use various style-changing HTML elements such as `<font>`, `<b>`, and `<i>` directly.

> *All ASP.NET developers should have a good understanding of HTML. For more information on HTML, please read Wrox's Beginning Web Programming with HTML, XHTML, and CSS (Wiley Publishing, Inc.; ISBN 978-0470-25931-3). You can also learn more about HTML and CSS design in ASP.NET by looking at Chapter 18 of this book.*

Using various HTML elements, you can change the appearance of many items contained on your pages. For instance, you can change a string's style as follows:

```
<font face="verdana"><b><i>Pork chops and applesauce</i></b></font>
```

You can go through an entire application and change the style of page elements using any of the appropriate HTML elements. You'll quickly find that this method works, but it is tough to maintain. To make any global style changes to your application, this method requires that you go through your application line-by-line to change each item individually. This can get cumbersome very fast!

Besides applying HTML elements to items to change their style, you can use another method known as *Cascading Style Sheets* (CSS). This alternative, but greatly preferred, styling technique allows you to assign formatting properties to HTML tags throughout your document in a couple of different ways. One way is to apply these styles directly to the HTML elements in your pages using *inline styles*. The other way involves placing these styles in an external stylesheet that can be placed either directly in an ASP.NET page or kept in a separate document that is simply referenced in the ASP.NET page. You explore these methods in the following sections.

Applying Styles Directly to HTML Elements

The first method of using CSS is to apply the styles directly to the tags contained in your ASP.NET pages. For instance, you apply a style to a string, as shown in Listing 2-3.

Listing 2-3: Applying CSS styles directly to HTML elements

```
<p style="color:blue; font-weight:bold">
   Pork chops and applesauce
</p>
```

This text string is changed by the CSS included in the `<p>` element so that the string appears bold and blue. Using the style attribute of the `<p>` element, you can change everything that appears between the opening and closing `<p>` elements. When the page is generated, the first style change applied is to the text between the `<p>` elements. In this example, the text has changed to the color blue because of the `color:blue` declaration, and then the `font-weight:bold` declaration is applied. You can separate the styling declarations using semicolons, and you can apply as many styles as you want to your elements.

Applying CSS styles in this manner presents the same problem as simply applying various HTML style elements — this is a tough structure to maintain. If styles are scattered throughout your pages, making global style changes can be rather time consuming. Putting all the styles together in a stylesheet is the best approach. A couple of methods can be used to build your stylesheets.

Working with the Visual Studio Style Builder

Visual Studio 2008 includes Style Builder, a tool that makes the building of CSS styles fairly simple. It can be quite a time saver because so many possible CSS definitions are available to you. If you are new to CSS, this tool can make all the difference.

The Visual Studio Style Builder enables you to apply CSS styles to individual elements or to construct your own stylesheets. To access the New Style tool when applying a style to a single page element, highlight the page element and then right-click it. From the menu that appears, select Style. Style Builder is shown in Figure 2-5.

You can use the Visual Studio Style Builder to change quite a bit about your selected item. After making all the changes you want and clicking OK, you see the styles you chose applied to the selected element.

Creating External StyleSheets

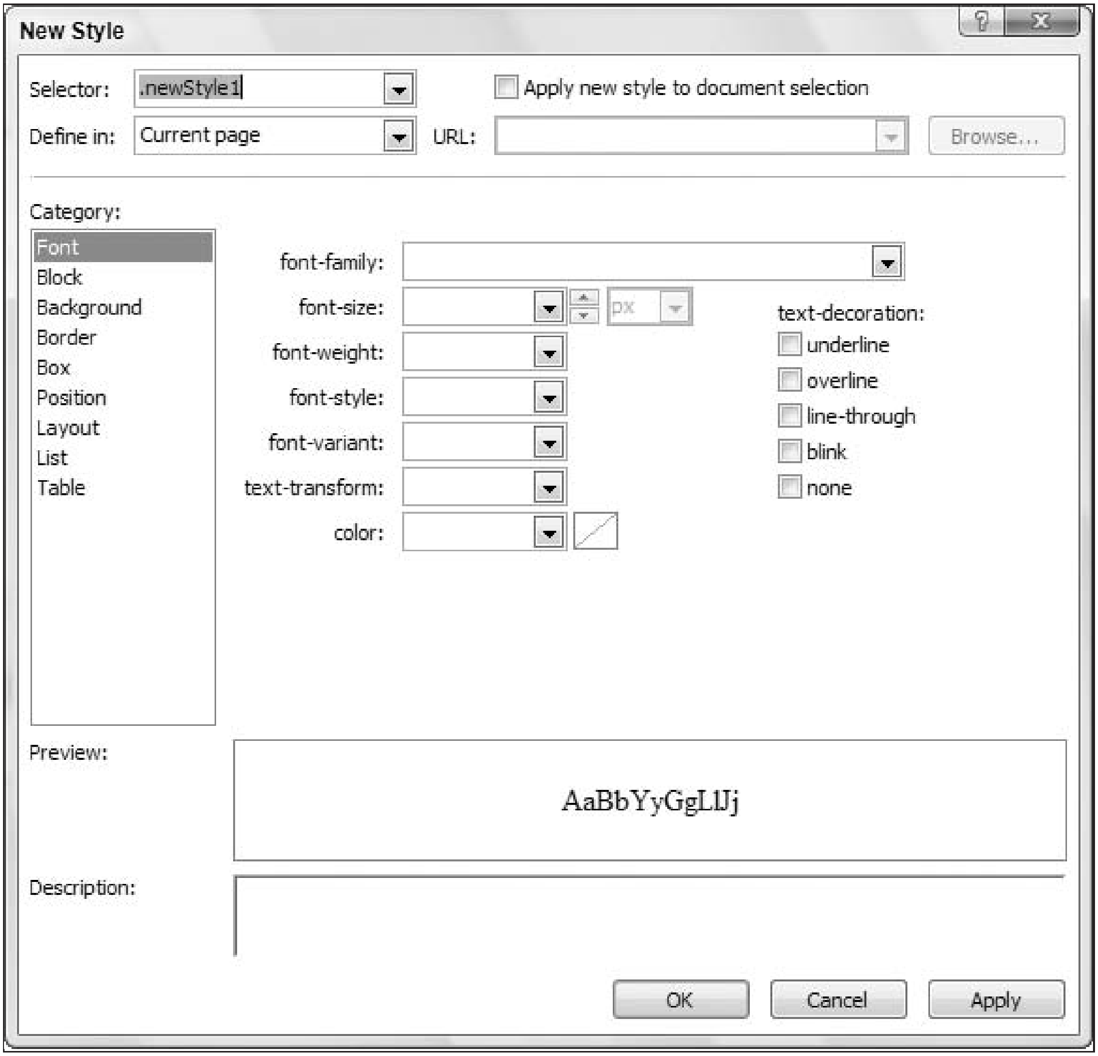
You can use a couple of different methods to create stylesheets. The most common method is to create an *external* stylesheet — a separate stylesheet file that is referenced in the pages that employ the defined styles. To begin the creation of your external stylesheet, add a new item to your project. From the Add New Item dialog box, create a stylesheet called `StyleSheet.css`. Add the file to your project by pressing the Add button. Figure 2-6 shows the result.

Using an external stylesheet within your application enables you to make global changes to the look-and-feel of your application quickly. Simply making a change at this central point cascades the change as defined by the stylesheet to your entire application.

Creating Internal Stylesheets

The second method for applying a stylesheet to a particular ASP.NET page is to bring the defined stylesheet into the actual document by creating an *internal* stylesheet. Instead of making a reference to an external stylesheet file, you bring the style definitions into the document. Note, however, that it is considered best practice to use external, rather than internal, stylesheets.

Consider using an internal stylesheet only if you are applying certain styles to a small number of pages within your application. Listing 2-4 shows the use of an internal stylesheet.



New Style

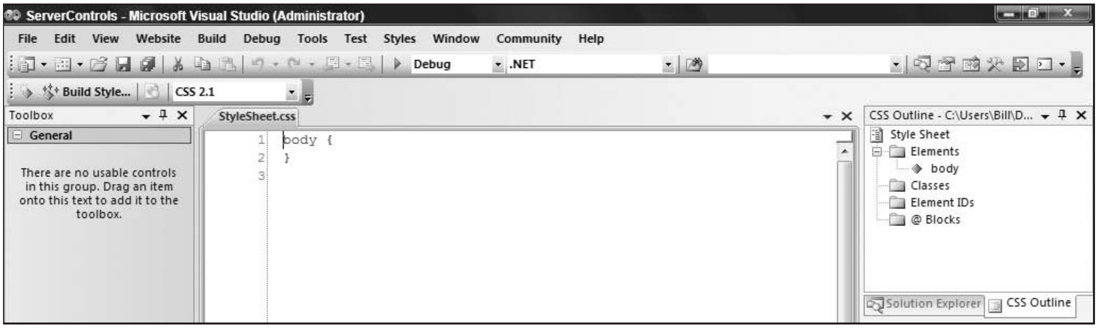
Selector: .newStyle1

Define in: Current page

☐ Apply new style to document selection

URL:
Browse...
Category:
Font
Block
Background
Border
Box
Position
Layout
List
Table
font-family:
font-size:
px
font-weight:
font-style:
font-variant:
text-transform:
color:
text-decoration:
☐ underline
☐ overline
☐ line-through
☐ blink
☐ none
Preview:
AaBbYyGgLlJj
Description:
OK
Cancel
Apply

Figure 2-5



ServerControls - Microsoft Visual Studio (Administrator)

StyleSheet.css

```
body {
}
```

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

File
Edit
View
Website
Build
Debug
Tools
Test
Styles
Window
Community
Help
Debug
.NET
Build Style...
CSS 2.1
Toolbox
General
CSS Outline - C:\Users\Bill\D...
Style Sheet
Elements
body
Classes
Element IDs
@ Blocks
Solution Explorer
CSS Outline

Figure 2-6

Listing 2-4: Using an internal stylesheet

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>My ASP.NET Page</title>

    <style type="text/css">
       <!--
          body {
             font-family: Verdana;
          }

          a:link {
             text-decoration: none;
             color: blue;
          }

          a:visited {
             text-decoration: none;
             color: blue;
          }

          a:hover {
             text-decoration: underline;
             color: red;
          }

       -->
    </style>

</head>
<body>
    <form id="form1" runat="server">
    <div>
        <a href="Default.aspx"<Home</a>
    </div>
    </form>
</body>
</html>
```

In this document, the internal stylesheet is set inside the opening and closing `<head>` elements. Although this is not a requirement, it is considered best practice. The stylesheet itself is placed between `<style>` tags with a type attribute defined as `text/css`.

HTML comment tags are included because not all browsers support internal stylesheets (it is generally the older browsers that do not accept them). Putting HTML comments around the style definitions hides these definitions from very old browsers. Except for the comment tags, the style definitions are handled in the same way they are done in an external stylesheet.

HTML Server Controls

ASP.NET enables you to take HTML elements and, with relatively little work on your part, turn them into server-side controls. Afterward, you can use them to control the behavior and actions of elements implemented in your ASP.NET pages.

Of course, you can place any HTML you want in your pages. You have the option of using the HTML placed in the page as a server-side control. You can also find a list of HTML elements contained in the Toolbox of Visual Studio (shown in Figure 2-7).

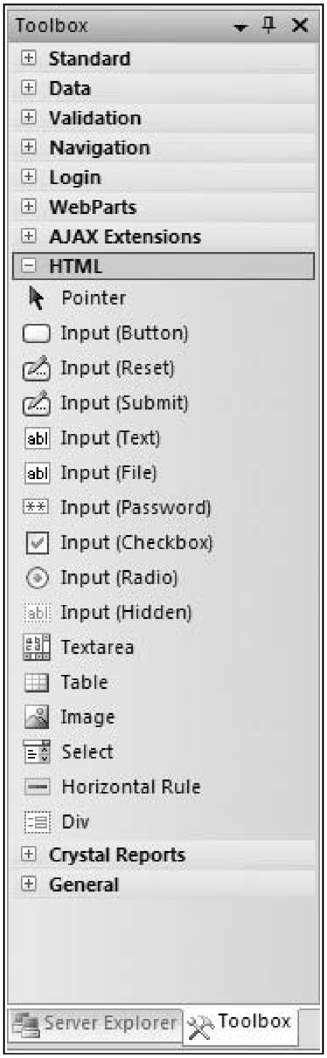
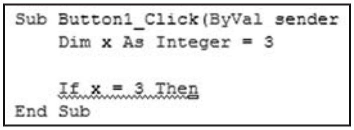

Toolbox
Standard
Data
Validation
Navigation
Login
WebParts
AJAX Extensions
HTML
Pointer
Input (Button)
Input (Reset)
Input (Submit)
Input (Text)
Input (File)
Input (Password)
Input (Checkbox)
Input (Radio)
Input (Hidden)
Textarea
Table
Image
Select
Horizontal Rule
Div
Crystal Reports
General
Server Explorer
Toolbox

Figure 2-7

Dragging and dropping any of these HTML elements from the Toolbox to the Design or Source view of your ASP.NET page in the Document window simply produces the appropriate HTML element. For instance, placing an HTML Button control on your page produces the following results in your code:

```
<input id="Button1" type="button" value="button" />
```

In this state, the Button control is not a server-side control. It is simply an HTML element and nothing more. You can turn this into an HTML server control in a couple of different ways. First let's take a look at how you would approach this if you were using Visual Studio 2005. From VS2005, in Design view, you can right-click the element and select Run As Server Control from the menu. This causes a few things to happen. The first thing is that a small green triangle appears on the visual element. The Button element, after it has been turned into an HTML server control, looks like Figure 2-8.



```
Sub Button1_Click(ByVal sender
    Dim x As Integer = 3

    If x = 3 Then
End Sub
```

Figure 2-8

In Source view, you simply change the HTML element by adding a `runat="server"` to the control:

```
<input id="Button1" type="button" value="button" runat="server" />
```

Using Visual Studio 2008, you won't find the Run As Server Control option in the menu. Therefore, in the Source view of the page, select the `Button1` option in the Object drop-down list on the page. At first, you will see only that `Button1` is available only in the client-side objects, as illustrated in Figure 2-9.

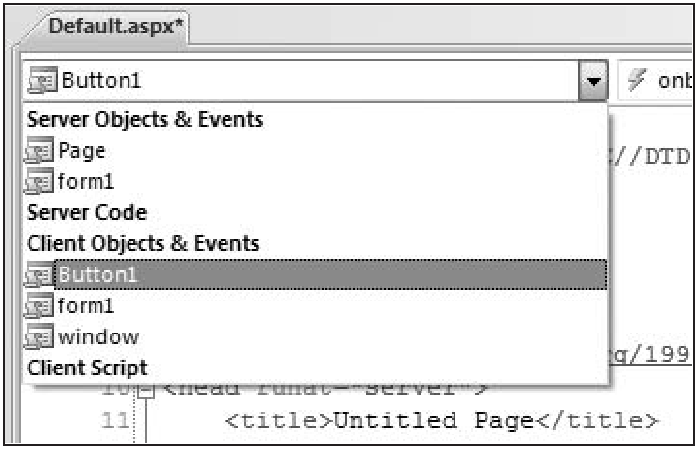


Figure 2-9

By adding the `runat="server"` to the element yourself, going back to this drop-down list, you will notice that the `Button1` object is now presented in the Server Objects & Events section in addition to the Client Objects & Events section, as shown in Figure 2-10.

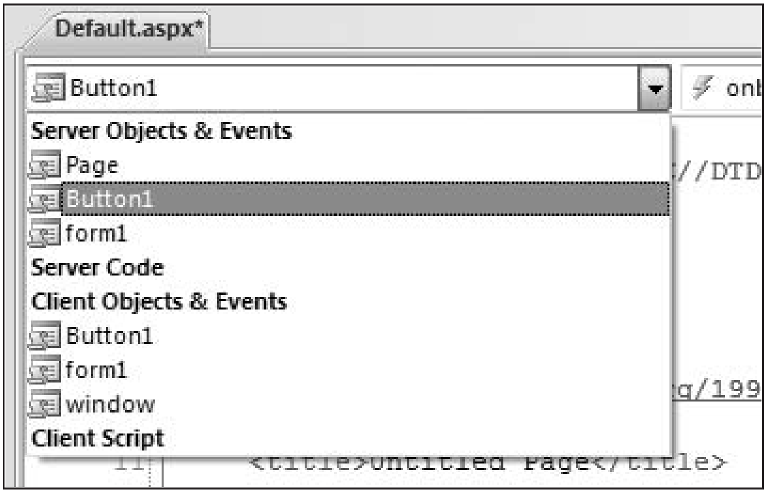

Default.aspx*
Button1
onl
Server Objects & Events
Page
Button1
form1
Server Code
Client Objects & Events
Button1
form1
window
Client Script
//DTD
g/199
<title>Untitled Page</title>

Figure 2-10

After the element is converted to a server control (through the addition of the `runat="server"` attribute and value), you can work with the selected element as you would work with any of the Web server controls. For instance, selecting `Button1` from the Source view of the page in the Server Objects & Events section and then selecting the `ServerClick` option from the list of server-side events generates a button-click event for the control. Listing 2-5 shows an example of some HTML server controls.

Listing 2-5: Working with HTML server controls

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_ServerClick(ByVal sender As Object, _
       ByVal e As System.EventArgs)
        Response.Write("Hello " & Text1.Value)
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Using HTML Server Controls</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        What is your name?<br />
        <input id="Text1" type="text" runat="server" />
        <input id="Button1" type="button" value="Submit" runat="server"
         onserverclick="Button1_ServerClick" />
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_ServerClick(object sender, EventArgs e)
    {
        Response.Write("Hello " + Text1.Value);
    }
</script>
```

In this example, you can see two HTML server controls on the page. Both are simply typical HTML elements with the additional `runat="server"` attribute added. If you are working with HTML elements as server controls, you must include an `id` attribute so that the server control can be identified in the server-side code.

The Button control includes a reference to a server-side event using the `OnServerClick` attribute. This attribute points to the server-side event that is triggered when an end user clicks the button — in this case, `Button1_ServerClick`. Within the `Button1_ServerClick` event, the value placed in the text box is output by using the `Value` property.

Looking at the HtmlControl Base Class

All the HTML server controls use a class that is derived from the `HtmlControl` base class (fully qualified as `System.Web.UI.HtmlControls.HtmlControl`). These classes expose many properties from the control's derived class. The following table details some of the properties available from this base class. Some of these items are themselves derived from the base `Control` class.

Method or Property	Description
`Attributes`	Provides a collection of name/value of all the available attributes specified in the control, including custom attributes.
`Disabled`	Allows you to get or set whether the control is disabled using a Boolean value.
`EnableTheming`	Enables you, using a Boolean value, to get or set whether the control takes part in the page theming capabilities.
`EnableViewState`	Allows you to get or set a Boolean value that indicates whether the control participates in the page's view state capabilities.
`ID`	Allows you to get or set the unique identifier for the control.
`Page`	Allows you to get a reference to the `Page` object that contains the specified server control.
`Parent`	Gets a reference to the parent control in the page control hierarchy.
`Site`	Provides information about the container for which the server control belongs.
`SkinID`	When the `EnableTheming` property is set to `True`, the `SkinID` property specifies the named skin that should be used in setting a theme.

Method or Property	Description
`Style`	Makes references to the CSS style collection that applies to the specified control.
`TagName`	Provides the name of the element that is generated from the specified control.
`Visible`	Specifies whether the control is visible (rendered) on the generated page.

You can find a more comprehensive list in the SDK.

Looking at the HtmlContainerControl Class

The `HtmlControl` base class is used for those HTML classes that are focused on HTML elements that can be contained within a single node. For instance, the `<img>`, `<input>`, and `<link>` elements work from classes derived from the `HtmlControl` class.

Other HTML elements such as `<a>`, `<form>`, and `<select>`, require an opening and closing set of tags. These elements use classes that are derived from the `HtmlContainerControl` class — a class specifically designed to work with HTML elements that require a closing tag.

Because the `HtmlContainerControl` class is derived from the `HtmlControl` class, you have all the `HtmlControl` class's properties and methods available to you as well as some new items that have been declared in the `HtmlContainerControl` class itself. The most important of these are the `InnerText` and `InnerHtml` properties:

- `InnerHtml`: Enables you to specify content that can include HTML elements to be placed between the opening and closing tags of the specified control.
- `InnerText`: Enables you to specify raw text to be placed between the opening and closing tags of the specified control.

Looking at All the HTML Classes

It is quite possible to work with every HTML element because a corresponding class is available for each one of them. The .NET Framework documentation shows the following classes for working with your HTML server controls:

- `HtmlAnchor` controls the `<a>` element.
- `HtmlButton` controls the `<button>` element.
- `HtmlForm` controls the `<form>` element.
- `HtmlHead` controls the `<head>` element.
- `HtmlImage` controls the `<img>` element.
- `HtmlInputButton` controls the `<input type="button">` element.
- `HtmlInputCheckBox` controls the `<input type="checkbox">` element.
- `HtmlInputFile` controls the `<input type="file">`element.

- ☐ `HtmlInputHidden` controls the `<input type="hidden">` element.
- ☐ `HtmlInputImage` controls the `<input type="image">` element.
- ☐ `HtmlInputPassword` controls the `<input type="password">` element.
- ☐ `HtmlInputRadioButton` controls the `<input type="radio">` element.
- ☐ `HtmlInputReset` controls the `<input type="reset">` element.
- ☐ `HtmlInputSubmit` controls the `<input type="submit">` element.
- ☐ `HtmlInputText` controls the `<input type="text">` element.
- ☐ `HtmlLink` controls the `<link>`element.
- ☐ `HtmlMeta` controls the `<meta>` element.
- ☐ `HtmlSelect` controls the `<select>` element.
- ☐ `HtmlTable` controls the `<table>` element.
- ☐ `HtmlTableCell` controls the `<td>` element.
- ☐ `HtmlTableRow` controls the `<tr>` element.
- ☐ `HtmlTextArea` controls the `<textarea>` element.
- ☐ `HtmlTitle` controls the `<title>` element.

You gain access to one of these classes when you convert an HTML element to an HTML server control. For example, convert the `<title>` element to a server control this way:

```
<title id="Title1" runat="Server"/>
```

That gives you access to the `HtmlTitle` class for this particular HTML element. Using this class instance, you can perform a number of tasks including providing a text value for the page title dynamically:

VB

```
Title1.Text = DateTime.Now.ToString()
```

C#

```
Title1.Text = DateTime.Now.ToString();
```

You can get most of the HTML elements you need by using these classes, but a considerable number of other HTML elements are at your disposal that are not explicitly covered by one of these HTML classes. For example, the `HtmlGenericControl` class provides server-side access to any HTML element you want.

Using the HtmlGenericControl Class

You should be aware of the importance of the `HtmlGenericControl` class; it gives you some capabilities that you do not get from any other server control offered by ASP.NET. For instance, using the `HtmlGenericControl` class, you can get server-side access to the `<meta>`, `<p>`, `<span>`, or other elements that would otherwise be unreachable.

Listing 2-6 shows you how to change the `<meta>` element in your page using the `HtmlGenericControl` class.

Listing 2-6: Changing the <meta> element using the HtmlGenericControl class

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Meta1.Attributes("Name") = "description"
        Meta1.Attributes("CONTENT") = "Generated on: " & DateTime.Now.ToString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Using the HtmlGenericControl class</title>
    <meta id="Meta1" runat="server" />
</head>
<body>
    <form id="form1" runat="server">
    <div>
        The rain in Spain stays mainly in the plains.
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Meta1.Attributes["Name"] = "description";
        Meta1.Attributes["CONTENT"] = "Generated on: " + DateTime.Now.ToString();
    }
</script>
```

In this example, the page's `<meta>` element is turned into an HTML server control with the addition of the `id` and `runat` attributes. Because the `HtmlGenericControl` class (which inherits from `HtmlControl`) can work with a wide range of HTML elements, you cannot assign values to HTML attributes in the same manner as you do when working with the other HTML classes (such as `HtmlInputButton`). You assign values to the attributes of an HTML element through the use of the `HtmlGenericControl` class's `Attributes` property, specifying the attribute you are working with as a string value.

The following is a partial result of running the example page:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <meta id="Meta1" Name="description"
    CONTENT="Generated on: 2/5/2008 2:42:52 PM"></meta>
   <title>Using the HtmlGenericControl class</title>
</head>
```

By using the `HtmlGenericControl` class, along with the other HTML classes, you can manipulate every element of your ASP.NET pages from your server-side code.

Manipulating Pages and Server Controls with JavaScript

Developers generally like to include some of their own custom JavaScript functions in their ASP.NET pages. You have a couple of ways to do this. The first is to apply JavaScript directly to the controls on your ASP.NET pages. For example, look at a simple Label server control, shown in Listing 2-7, which displays the current date and time.

Listing 2-7: Showing the current date and time

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   TextBox1.Text = DateTime.Now.ToString()
End Sub
```

C#

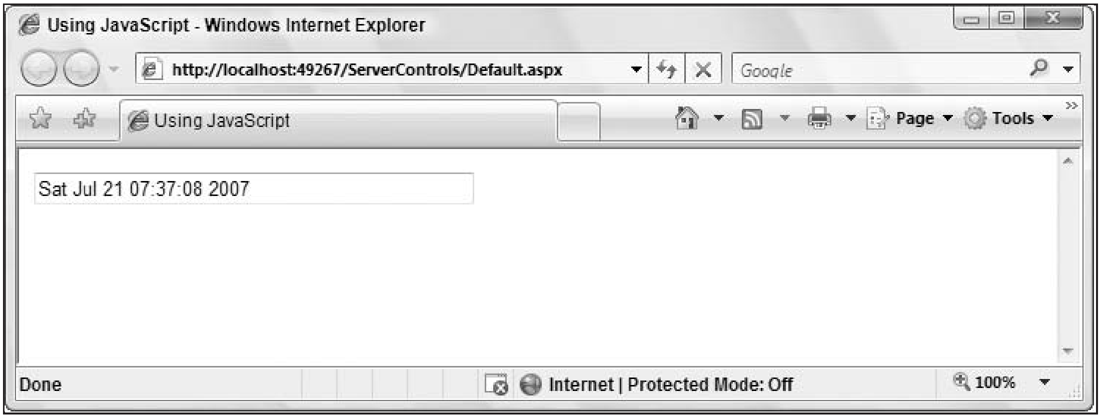
```
protected void Page_Load(object sender, EventArgs e) {
   TextBox1.Text = DateTime.Now.ToString();
}
```

This little bit of code displays the current date and time on the page of the end user. The problem is that the date and time displayed are correct for the Web server that generated the page. If someone sits in the Pacific time zone (PST), and the Web server is in the Eastern time zone (EST), the page won't be correct for that viewer. If you want the time to be correct for anyone visiting the site, regardless of where they reside in the world, you can employ JavaScript to work with the TextBox control, as illustrated in Listing 2-8.

Listing 2-8: Using JavaScript to show the current time for the end user

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Using JavaScript</title>
</head>
<body onload="javascript:document.forms[0]['TextBox1'].value=Date();">
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" Runat="server" Width="300"></asp:TextBox>
    </div>
    </form>
</body>
</html>
```

In this example, even though you are using a standard TextBox server control from the Web server control family, you can get at this control using JavaScript that is planted in the `onload` attribute of the `<body>` element. The value of the `onload` attribute actually points to the specific server control via an anonymous function by using the value of the `ID` attribute from the server control: `TextBox1`. You can get at other server controls on your page by employing the same methods. This bit of code produces the result illustrated in Figure 2-11.


Using JavaScript - Windows Internet Explorer
http://localhost:49267/ServerControls/Default.aspx
Google
Using JavaScript
Page
Tools
Done
Internet | Protected Mode: Off
100%

Sat Jul 21 07:37:08 2007

Figure 2-11

ASP.NET uses the `Page.ClientScript` property to register and place JavaScript functions on your ASP .NET pages. Three of these methods are reviewed here. More methods and properties than just these two are available through the `ClientScript` object (which references an instance of `System.Web.UI.ClientScriptManager`), but these are the more useful ones. You can find the rest in the SDK documentation.

> *The* `Page.RegisterStartupScript` *and the* `Page.RegisterClientScriptBlock` *methods from the .NET Framework 1.0/1.1 are now considered obsolete. Both of these possibilities for registering scripts required a key/script set of parameters. Because two separate methods were involved, there was an extreme possibility that some key name collisions would occur. The* `Page.ClientScript` *property is meant to bring all the script registrations under one umbrella, making your code less error prone.*

Using Page.ClientScript.RegisterClientScriptBlock

The `RegisterClientScriptBlock` method allows you to place a JavaScript function at the top of the page. This means that the script is in place for the startup of the page in the browser. Its use is illustrated in Listing 2-9.

Listing 2-9: Using the RegisterClientScriptBlock method

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
      Dim myScript As String = "function AlertHello() { alert('Hello ASP.NET'); }"
      Page.ClientScript.RegisterClientScriptBlock(Me.GetType(), "MyScript", _
         myScript, True)
    End Sub
```

```
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Adding JavaScript</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Button ID="Button1" Runat="server" Text="Button"
         OnClientClick="AlertHello()" />
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
      string myScript = @"function AlertHello() { alert('Hello ASP.NET'); }";
      Page.ClientScript.RegisterClientScriptBlock(this.GetType(),
         "MyScript", myScript, true);
    }
</script>
```

From this example, you can see that you create the JavaScript function `AlertHello()` as a string called `myScript`. Then using the `Page.ClientScript.RegisterClientScriptBlock` method, you program the script to be placed on the page. The two possible constructions of the `RegisterClientScriptBlock` method are the following:

- ❑ `RegisterClientScriptBlock` (*type*, *key*, *script*)
- ❑ `RegisterClientScriptBlock` (*type*, *key*, *script*, *script tag specification*)

In the example from Listing 2-9, you are specifying the type as `Me.GetType()`, the key, the script to include, and then a `Boolean` value setting of `True` so that .NET places the script on the ASP.NET page with `<script>` tags automatically. When running the page, you can view the source code for the page to see the results:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
    Adding JavaScript
</title></head>
<body>
    <form method="post" action="JavaScriptPage.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE"
 value="/wEPDwUKMTY3NzE5MjIyMGRkiyYSRMg+bcXi9DiawYlbxndiTDo=" />
</div>
```

```
<script type="text/javascript">
<!--
function AlertHello() { alert('Hello ASP.NET'); }// -->
</script>

    <div>
        <input type="submit" name="Button1" value="Button" onclick="AlertHello();"
         id="Button1" />
    </div>
    </form>
</body>
</html>
```

From this, you can see that the script specified was indeed included on the ASP.NET page before the page code. Not only were the `<script>` tags included, but the proper comment tags were added around the script (so older browsers will not break).

Using Page.ClientScript.RegisterStartupScript

The `RegisterStartupScript` method is not too much different from the `RegisterClientScriptBlock` method. The big difference is that the `RegisterStartupScript` places the script at the bottom of the ASP.NET page instead of at the top. In fact, the `RegisterStartupScript` method even takes the same constructors as the `RegisterClientScriptBlock` method:

- ❑ `RegisterStartupScript` (*type*, *key*, *script*)
- ❑ `RegisterStartupScript` (type, key, script, script tag specification)

So what difference does it make where the script is registered on the page? A lot, actually!

If you have a bit of JavaScript that is working with one of the controls on your page, in most cases you want to use the `RegisterStartupScript` method instead of `RegisterClientScriptBlock`. For example, you'd use the following code to create a page that includes a simple `<asp:TextBox>` control that contains a default value of `Hello ASP.NET`.

```
<asp:TextBox ID="TextBox1" Runat="server">Hello ASP.NET</asp:TextBox>
```

Then use the `RegisterClientScriptBlock` method to place a script on the page that utilizes the value in the `TextBox1` control, as illustrated in Listing 2-10.

Listing 2-10: Improperly using the RegisterClientScriptBlock method

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   Dim myScript As String = "alert(document.forms[0]['TextBox1'].value);"
   Page.ClientScript.RegisterClientScriptBlock(Me.GetType(), "myKey", myScript, _
      True)
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
   string myScript = @"alert(document.forms[0]['TextBox1'].value);";
   Page.ClientScript.RegisterClientScriptBlock(this.GetType(),
      "MyScript", myScript, true);
}
```

Running this page (depending on the version of IE your are using) gives you a JavaScript error, as shown in Figure 2-12.

Toolbox
Standard
Data
Validation
Navigation
Login
WebParts
HTML
General

Figure 2-12

The reason for the error is that the JavaScript function fired before the text box was even placed on the screen. Therefore, the JavaScript function did not find `TextBox1`, and that caused an error to be thrown by the page. Now try the `RegisterStartupScript` method shown in Listing 2-11.

Listing 2-11: Using the RegisterStartupScript method

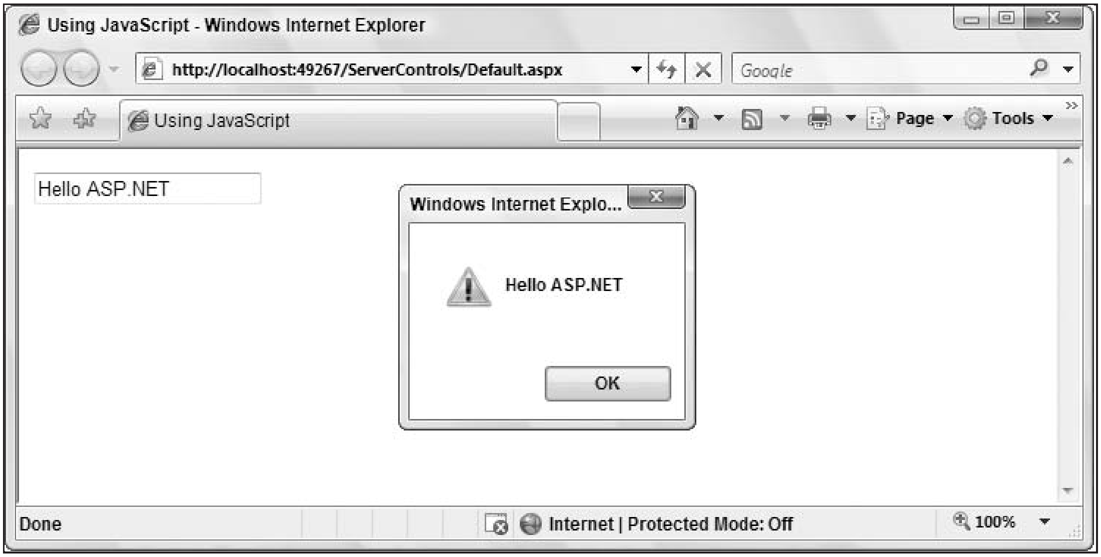
VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   Dim myScript As String = "alert(document.forms[0]['TextBox1'].value);"
   Page.ClientScript.RegisterStartupScript(Me.GetType(), "myKey", myScript, _
      True)
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
   string myScript = @"alert(document.forms[0]['TextBox1'].value);";
   Page.ClientScript.RegisterStartupScript(this.GetType(),
      "MyScript", myScript, true);
}
```

This approach puts the JavaScript function at the bottom of the ASP.NET page, so when the JavaScript actually starts, it finds the `TextBox1` element and works as planned. The result is shown in Figure 2-13.


Using JavaScript - Windows Internet Explorer
http://localhost:49267/ServerControls/Default.aspx
Google
Using JavaScript
Page
Tools
Windows Internet Explo...
OK
Done
Internet | Protected Mode: Off
100%

Hello ASP.NET

Hello ASP.NET

Figure 2-13

Using Page.ClientScript.RegisterClientScriptInclude

The final method is `RegisterClientScriptInclude`. Many developers place their JavaScript inside a `.js` file, which is considered a best practice because it makes it very easy to make global JavaScript changes to the application. You can register the script files on your ASP.NET pages using the `RegisterClientScriptInclude` method illustrated in Listing 2-12.

Listing 2-12: Using the RegisterClientScriptInclude method

VB

```
Dim myScript As String = "myJavaScriptCode.js"
Page.ClientScript.RegisterClientScriptInclude("myKey", myScript)
```

C#

```
string myScript = "myJavaScriptCode.js";
Page.ClientScript.RegisterClientScriptInclude("myKey", myScript);
```

This creates the following construction on the ASP.NET page:

```
<script src="myJavaScriptCode.js" type="text/javascript"></script>
```

Client-Side Callback

ASP.NET 3.5 includes a client callback feature that enables you to retrieve page values and populate them to an already-generated page without regenerating the page. This was introduced with ASP.NET 2.0. This capability makes it possible to change values on a page without going through the entire postback cycle; that means you can update your pages without completely redrawing the page. End users will not see the page flicker and reposition, and the pages will have a flow more like the flow of a thick-client application.

To work with the new callback capability, you have to know a little about working with JavaScript. This book does not attempt to teach you JavaScript. If you need to get up to speed on this rather large topic, check out Wrox's *Beginning JavaScript, Third Edition*, by Paul Wilton and Jeremy McPeak (Wiley Publishing, Inc., ISBN: 978-0-470-05151-1).

> *You can also accomplish client callbacks in a different manner using ASP.NET AJAX. You will find more information on this in Chapters 19 and 20.*

Comparing a Typical Postback to a Callback

Before you jump into some examples of the new callback feature, first look at a comparison to the current postback feature of a typical ASP.NET page.

When a page event is triggered on an ASP.NET page that is working with a typical postback scenario, a lot is going on. The diagram in Figure 2-14 illustrates the process.

In a normal postback situation, an event of some kind triggers an HTTP Post request to be sent to the Web server. An example of such an event might be the end user clicking a button on the form. This sends the HTTP Post request to the Web server, which then processes the request with the `IPostbackEventHandler` and runs the request through a series of page events. These events include loading the state (as found in the view state of the page), processing data, processing postback events, and finally rendering the page to be interpreted by the consuming browser once again. The process completely reloads the page in the browser, which is what causes the flicker and the realignment to the top of the page.

On the other hand, you have the alternative of using the callback capabilities, as shown in the diagram in Figure 2-15.

In this case, an event (again, such as a button click) causes the event to be posted to a script event handler (a JavaScript function) that sends off an asynchronous request to the Web server for processing. `ICallbackEventHandler` runs the request through a pipeline similar to what is used with the postback — but you notice that some of the larger steps (such as rendering the page) are excluded from the process chain. After the information is loaded, the result is returned to the script callback object. The script code then pushes this data into the Web page using JavaScript's capabilities to do this without refreshing the page. To understand how this all works, look at the simple example in the following section.

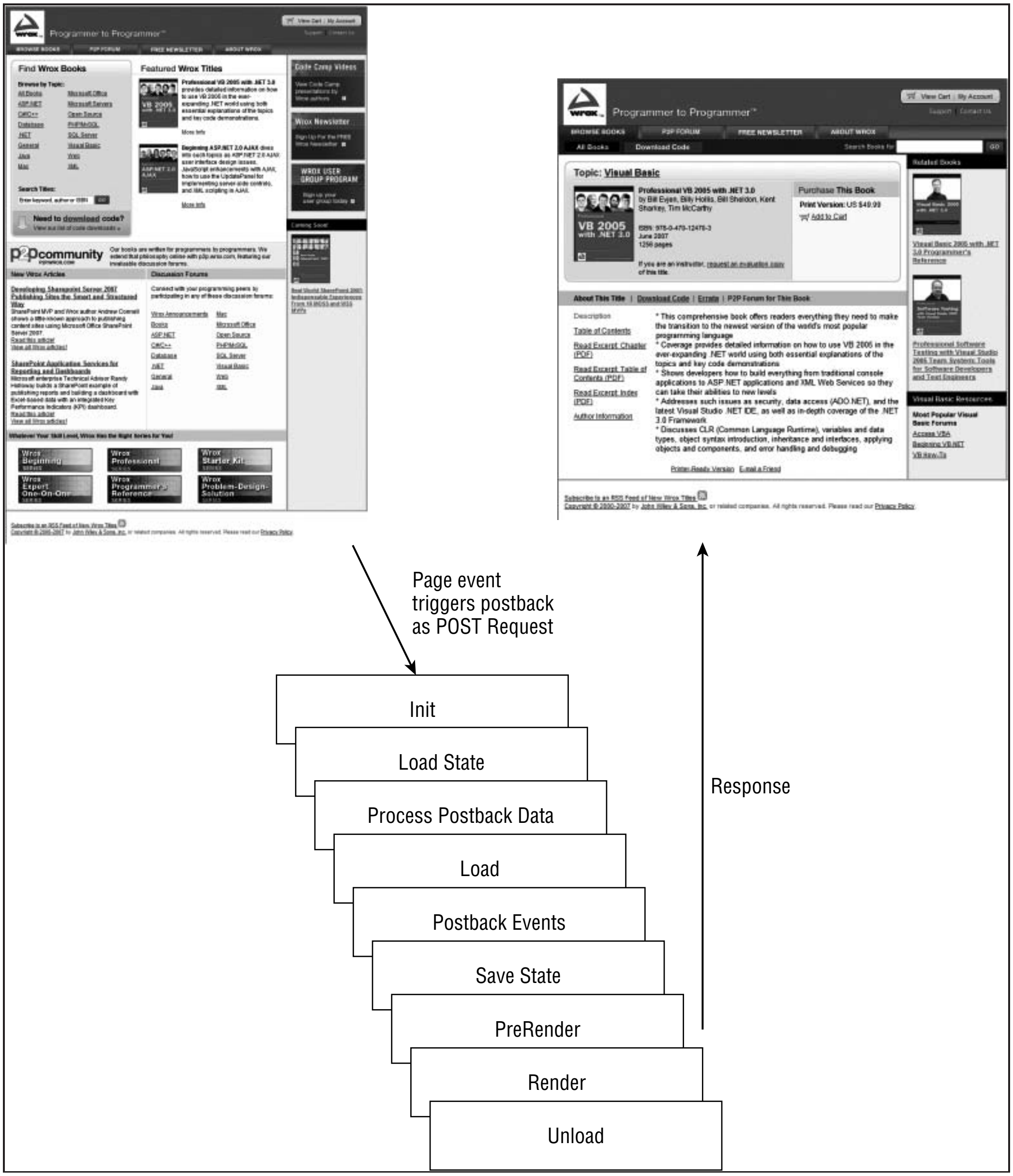

Page event
triggers postback
as POST Request
Init
Load State
Process Postback Data
Load
Postback Events
Save State
PreRender
Render
Unload
Response

Figure 2-14

Using the Callback Feature — A Simple Approach

Begin examining the callback feature by looking at how a simple ASP.NET page uses it. For this example, you have only an HTML button control and a TextBox server control (the Web server control version). The idea is that when the end user clicks the button on the form, the callback service is initiated and a random number is populated into the text box. Listing 2-13 shows an example of this in action.

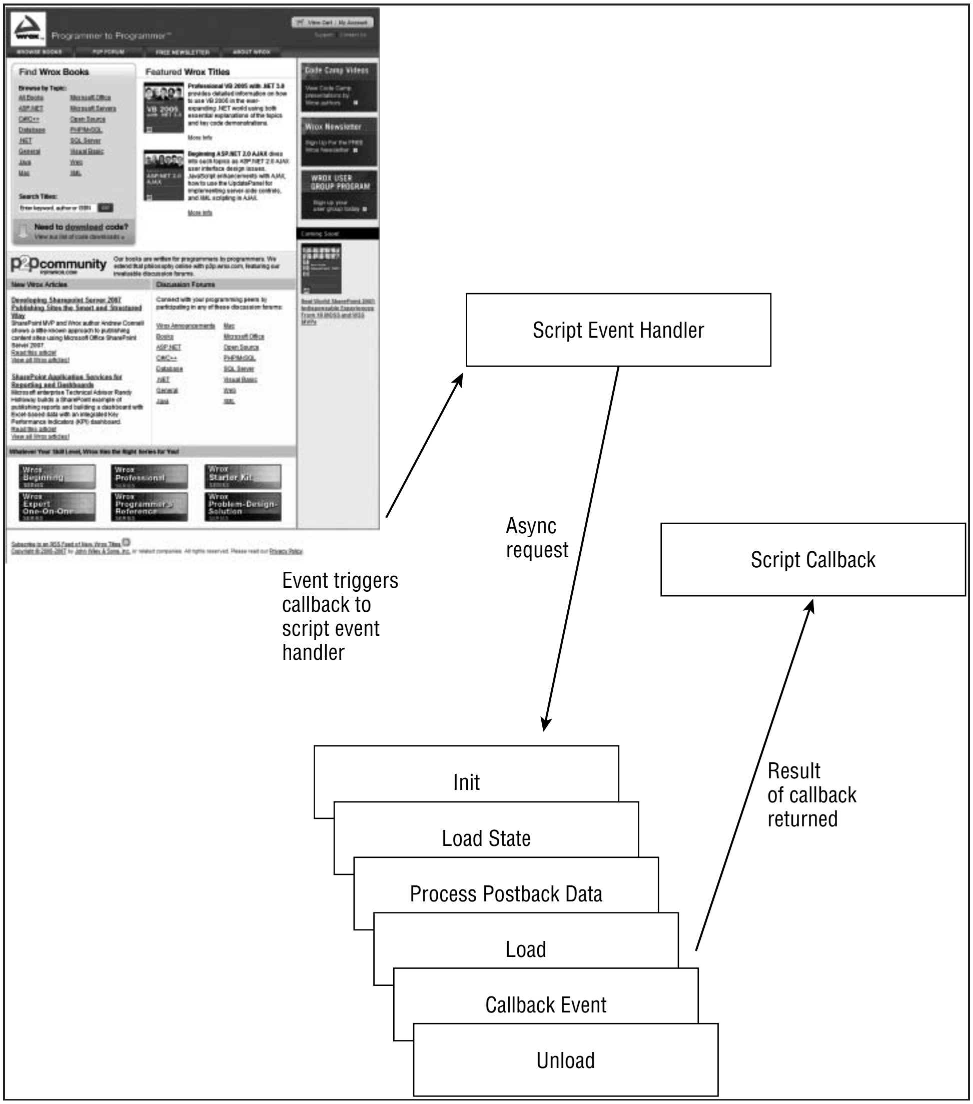

Script Event Handler
Async request
Event triggers callback to script event handler
Script Callback
Init
Load State
Process Postback Data
Load
Callback Event
Unload
Result of callback returned

Figure 2-15

Listing 2-13: Using the callback feature to populate a random value to a Web page

.aspx page (VB version)

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="RandomNumber.aspx.vb"
    Inherits="RandomNumber" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server"<
    <title>Callback Page</title>
    <script type="text/javascript">
        function GetNumber(){
            UseCallback();
        }

        function GetRandomNumberFromServer(TextBox1, context){
            document.forms[0].TextBox1.value = TextBox1;
        }
    </script>

</head>
<body>
    <form id="form1" runat="server">
    <div>
        <input id="Button1" type="button" value="Get Random Number"
         onclick="GetNumber()" />
        <br />
        <br />
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
    </div>
    </form>
</body>
</html>
```

VB (code-behind)

```
Partial Class RandomNumber
    Inherits System.Web.UI.Page
    Implements System.Web.UI.ICallbackEventHandler

    Dim _callbackResult As String = Nothing

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
       Handles Me.Load

        Dim cbReference As String = Page.ClientScript.GetCallbackEventReference( _
           Me, "arg", "GetRandomNumberFromServer", "context")
        Dim cbScript As String = "function UseCallback(arg, context)" & _
           "{" & cbReference & ";" & "}"

        Page.ClientScript.RegisterClientScriptBlock(Me.GetType(), _
           "UseCallback", cbScript, True)
    End Sub

    Public Sub RaiseCallbackEvent(ByVal eventArgument As String) _
       Implements System.Web.UI.ICallbackEventHandler.RaiseCallbackEvent

       _callbackResult = Rnd().ToString()
    End Sub

    Public Function GetCallbackResult() As String _
       Implements System.Web.UI.ICallbackEventHandler.GetCallbackResult

        Return _callbackResult
    End Function
End Class
```

C# (code-behind)

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class RandomNumber : System.Web.UI.Page,
   System.Web.UI.ICallbackEventHandler
{
    private string _callbackResult = null;

    protected void Page_Load(object sender, EventArgs e)
    {
        string cbReference = Page.ClientScript.GetCallbackEventReference(this,
           "arg", "GetRandomNumberFromServer", "context");
        string cbScript = "function UseCallback(arg, context)" +
           "{" + cbReference + ";" + "}";

        Page.ClientScript.RegisterClientScriptBlock(this.GetType(),
           "UseCallback", cbScript, true);
    }

    public void RaiseCallbackEvent(string eventArg)
    {
        Random rnd = new Random();
        _callbackResult = rnd.Next().ToString();
    }

    public string GetCallbackResult()
    {
        return _callbackResult;
    }
}
```

When this page is built and run in the browser, you get the results shown in Figure 2-16.

Clicking the button on the page invokes the client callback capabilities of the page, and the page then makes an asynchronous request to the code behind of the same page. After getting a response from this part of the page, the client script takes the retrieved value and places it inside the text box — all without doing a page refresh!

Now take a look at the `.aspx` page, which simply contains an HTML button control and a TextBox server control. Notice that a standard HTML button control is used because a typical `<asp:button>` control does not work here. No worries. When you work with the HTML button control, just be sure to include an `onclick` event to point to the JavaScript function that initiates this entire process:

```
<input id="Button1" type="button" value="Get Random Number"
 onclick="GetNumber()" />
```

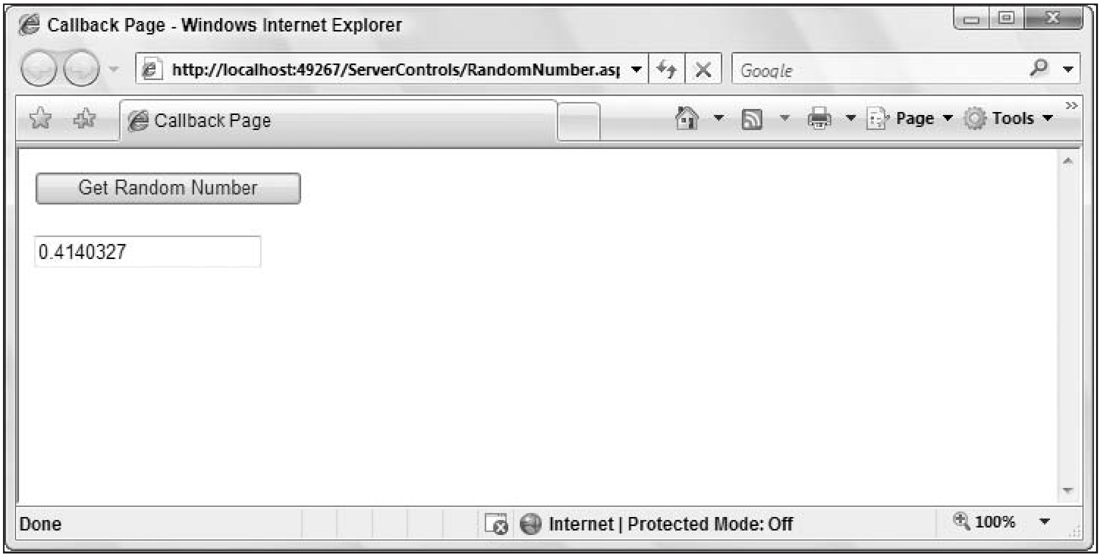

Callback Page - Windows Internet Explorer
http://localhost:49267/ServerControls/RandomNumber.as|
Google
Callback Page
Page
Tools
Get Random Number
0.4140327
Done
Internet | Protected Mode: Off
100%

Figure 2-16

You do not have to do anything else with the controls themselves. The final thing to include in the page is the client-side JavaScript functions to take care of the callback to the server-side functions. `GetNumber()` is the first JavaScript function that's instantiated. It starts the entire process by calling the name of the client script handler that is defined in the page's code behind. A `string` type result from `GetNumber()` is retrieved using the `GetRandomNumberFromServer()` function. `GetRandomNumberFromServer()` simply populates the `string` value retrieved and makes that the value of the Textbox control — specified by the value of the `ID` attribute of the server control (`TextBox1`):

```
<script type="text/javascript">
   function GetNumber(){
      UseCallback();
   }

   function GetRandomNumberFromServer(TextBox1, context){
      document.forms[0].TextBox1.value = TextBox1;
   }
</script>
```

Now turn your attention to the code behind.

The `Page` class of the Web page implements the `System.Web.UI.ICallbackEventHandler` interface:

```
Partial Class RandomNumber
    Inherits System.Web.UI.Page
    Implements System.Web.UI.ICallbackEventHandler

    ' Code here

End Class
```

This interface requires you to implement a couple of methods — the `RaiseCallbackEvent` and the `GetCallbackResult` methods, both of which work with the client script request. `RaiseCallbackEvent` enables you to do the work of retrieving the value from the page, but the value can be only of type `string`:

```
Public Sub RaiseCallbackEvent(ByVal eventArgument As String) _
   Implements System.Web.UI.ICallbackEventHandler.RaiseCallbackEvent

   _callbackResult = Rnd().ToString()
End Sub
```

The `GetCallbackResult` is the method that actually grabs the returned value to be used:

```
Public Function GetCallbackResult() As String _
   Implements System.Web.UI.ICallbackEventHandler.GetCallbackResult

        Return _callbackResult
End Function
```

In addition, the `Page_Load` event includes the creation and placement of the client callback script manager (the function that will manage requests and responses) on the client:

```
Dim cbReference As String = Page.GetCallbackEventReference(Me, "arg", _
   "GetRandomNumberFromServer", "context")
Dim cbScript As String = "function UseCallback(arg, context)" & _
   "{" & cbReference & ";" & "}"

Page.ClientScript.RegisterClientScriptBlock(Me.GetType(), _
   "UseCallback", cbScript, True)
```

The function placed on the client for the callback capabilities is called `UseCallback()`. This `string` is then populated to the Web page itself using the `Page.ClientScript.RegisterClientScripBlock` that also puts `<script>` tags around the function on the page. Make sure that the name you use here is the same name you use in the client-side JavaScript function presented earlier.

In the end, you have a page that refreshes content without refreshing the overall page. This opens the door to a completely new area of possibilities. One caveat is that the callback capabilities described here use XmlHTTP and, therefore, the client browser needs to support XmlHTTP (Microsoft's Internet Explorer and FireFox do support this feature). Because of this, .NET Framework 2.0 and 3.5 have the `SupportsCallBack` and the `SupportsXmlHttp` properties. To ensure this support, you could put a check in the page's code behind when the initial page is being generated. It might look similar to the following:

VB

```
If (Page.Request.Browser.SupportsXmlHTTP) Then

End If
```

C#

```
if (Page.Request.Browser.SupportsXmlHTTP == true) {

}
```

Using the Callback Feature with a Single Parameter

Now you will build a Web page that utilizes the callback feature but requires a parameter to retrieve a returned value. At the top of the page, place a text box that gathers input from the end user, a button, and another text box to populate the page with the result from the callback.

The page asks for a ZIP Code from the user and then uses the callback feature to instantiate an XML Web service request on the server. The Web service returns the latest weather for that particular ZIP Code in a string format. Listing 2-14 shows an example of the page.

Listing 2-14: Using the callback feature with a Web service

.aspx page (VB version)

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="WSCallback.aspx.vb"
    Inherits="WSCallback" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Web Service Callback</title>

    <script type="text/javascript">
        function GetTemp(){
            var zipcode = document.forms[0].TextBox1.value;
            UseCallback(zipcode, "");
        }

        function GetTempFromServer(TextBox2, context){
            document.forms[0].TextBox2.value = "Zipcode: " +
            document.forms[0].TextBox1.value + " | Temp: " + TextBox2;
        }
    </script>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
        <br />
        <input id="Button1" type="button" value="Get Temp" onclick="GetTemp()" />
        <br />
        <asp:TextBox ID="TextBox2" Runat="server" Width="400px">
        </asp:TextBox>
        <br />
        <br />
    </div>
    </form>
</body>
</html>
```

VB (code-behind)

```
Partial Class WSCallback
    Inherits System.Web.UI.Page
    Implements System.Web.UI.IcallbackEventHandler

    Dim _callbackResult As String = Nothing

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
       Handles Me.Load

        Dim cbReference As String = Page.ClientScript.GetCallbackEventReference( _
           Me, "arg", "GetTempFromServer", "context")
        Dim cbScript As String = "function UseCallback(arg, context)" & _
           "{" & cbReference & ";" & "}"

        Page.ClientScript.RegisterClientScriptBlock(Me.GetType(), _
           "UseCallback", cbScript, True)
    End Sub

    Public Sub RaiseCallbackEvent(ByVal eventArgument As String) _
       Implements System.Web.UI.ICallbackEventHandler.RaiseCallbackEvent

        Dim ws As New Weather.TemperatureService
        _callbackResult = ws.getTemp(eventArgument).ToString()
    End Sub
    Public Function GetCallbackResult() As String _
       Implements System.Web.UI.ICallbackEventHandler.GetCallbackResult

        Return _callbackResult
    End Function
End Class
```

C# (code-behind)

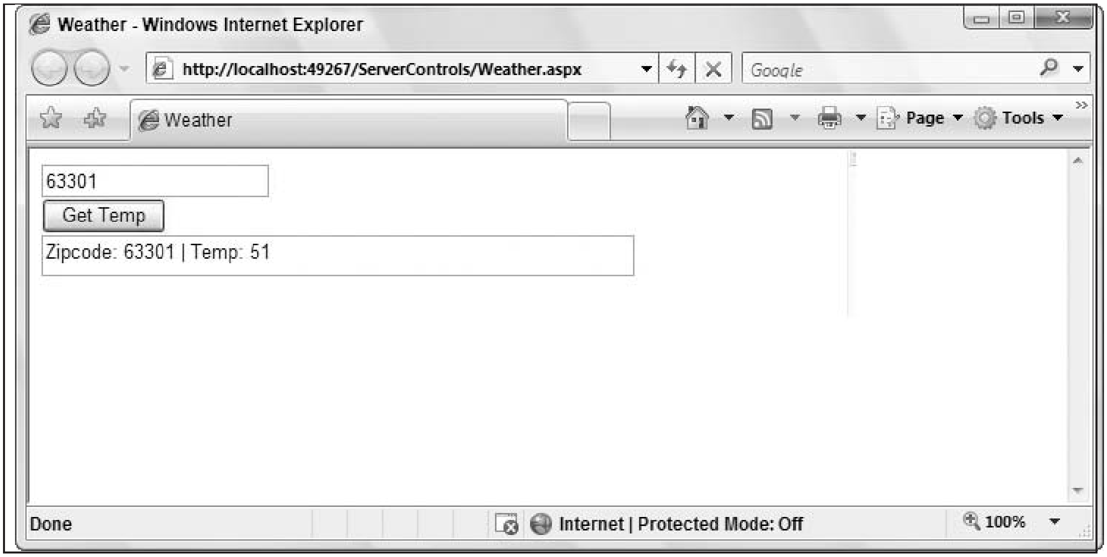
```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class WSCallback : System.Web.UI.Page,
   System.Web.UI.ICallbackEventHandler
{
    private string _callbackResult = null;

    protected void Page_Load(object sender, EventArgs e)
    {
        string cbReference = Page.ClientScript.GetCallbackEventReference(this,
           "arg", "GetTempFromServer", "context");
```

```
            string cbScript = "function UseCallback(arg, context)" +
               "{" + cbReference + ";" + "}";

            Page.ClientScript.RegisterClientScriptBlock(this.GetType(),
               "UseCallback", cbScript, true);
        }

        public void RaiseCallbackEvent(string eventArg)
        {
            Weather.TemperatureService ws = new Weather.TemperatureService();
            _callbackResult = ws.getTemp(eventArg).ToString();
        }

        public string GetCallbackResult()
        {
            return _callbackResult;
        }
    }
```

What you do not see on this page from the listing is that a Web reference has been made to a remote Web service that returns the latest weather to the application based on a ZIP Code the user supplied.

> *To get at the Web service used in this demo, the location of the WSDL file at the time of this writing is* `http://ws.strikeiron.com/InnerGears/ForecastByZip2?WSDL`. *For more information on working with Web services in your ASP.NET applications, check out Chapter 30.*

After building and running this page, you get the results illustrated in Figure 2-17.


Weather - Windows Internet Explorer
http://localhost:49267/ServerControls/Weather.aspx
Google
Weather
Page
Tools
Done
Internet | Protected Mode: Off
100%

63301

Get Temp

Zipcode: 63301 | Temp: 51

Figure 2-17

The big difference with the client callback feature is that this example sends in a required parameter. That is done in the `GetTemp()` JavaScript function on the `.aspx` part of the page:

```
function GetTemp(){
   var zipcode = document.forms[0].TextBox1.value;
   UseCallback(zipcode, "");
}
```

The JavaScript function shows the population that the end user input into `TextBox1` and places its value in a variable called `zipcode` that is sent as a parameter in the `UseCallback()` method.

This example, like the previous one, updates the page without doing a complete page refresh.

Using the Callback Feature — A More Complex Example

So far, you have seen an example of using the callback feature to pull back a single item as well as to pull back a string whose output is based on a single parameter that was passed to the engine. The next example takes this operation one step further and pulls back a collection of results based upon a parameter provided.

This example works with an instance of the Northwind database found in SQL Server. For this example, create a single page that includes a TextBox server control and a button. Below that, place a table that will be populated with the customer details from the customer ID provided in the text box. The `.aspx` page for this example is provided in Listing 2-15.

Listing 2-15: An ASP.NET page to collect the CustomerID from the end user

.aspx Page

```
<%@ Page Language="VB" AutoEventWireup="false"
    CodeFile="Default.aspx.vb" Inherits="_Default" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Customer Details</title>

    <script type="text/javascript">
        function GetCustomer(){
            var customerCode = document.forms[0].TextBox1.value;
            UseCallback(customerCode, "");
        }

        function GetCustDetailsFromServer(result, context){
            var i = result.split("|");
            customerID.innerHTML = i[0];
            companyName.innerHTML = i[1];
            contactName.innerHTML = i[2];
            contactTitle.innerHTML = i[3];
            address.innerHTML = i[4];
            city.innerHTML = i[5];
            region.innerHTML = i[6];
            postalCode.innerHTML = i[7];
            country.innerHTML = i[8];
            phone.innerHTML = i[9];
            fax.innerHTML = i[10];
        }
    </script>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>&nbsp;
        <input id="Button1" type="button" value="Get Customer Details"
         onclick="GetCustomer()" /><br />
        <br />
      <table cellspacing="0" cellpadding="4" rules="all" border="1"
       id="DetailsView1"
       style="background-color:White;border-color:#3366CC;border-width:1px;
          border-style:None;height:50px;width:400px;border-collapse:collapse;">
        <tr style="color:#003399;background-color:White;">
           <td>CustomerID</td><td><span id="customerID" /></td>
        </tr><tr style="color:#003399;background-color:White;">
           <td>CompanyName</td><td><span id="companyName" /></td>
        </tr><tr style="color:#003399;background-color:White;">
           <td>ContactName</td><td><span id="contactName" /></td>
        </tr><tr style="color:#003399;background-color:White;">
           <td>ContactTitle</td><td><span id="contactTitle" /></td>
        </tr><tr style="color:#003399;background-color:White;">
           <td>Address</td><td><span id="address" /></td>
        </tr><tr style="color:#003399;background-color:White;">
           <td>City</td><td><span id="city" /></td>
        </tr><tr style="color:#003399;background-color:White;">
           <td>Region</td><td><span id="region" /></td>
        </tr><tr style="color:#003399;background-color:White;">
           <td>PostalCode</td><td><span id="postalCode" /></td>
        </tr><tr style="color:#003399;background-color:White;">
           <td>Country</td><td><span id="country" /></td>
        </tr><tr style="color:#003399;background-color:White;">
           <td>Phone</td><td><span id="phone" /></td>
        </tr><tr style="color:#003399;background-color:White;">
           <td>Fax</td><td><span id="fax" /></td>
        </tr>
      </table>
    </div>
    </form>
</body>
</html>
```

As in the previous examples, two JavaScript functions are contained in the page. The first, `GetCustomer()`, is the function that passes in the parameter to be processed by the code-behind file on the application server. This is quite similar to what appeared in the previous example.

The second JavaScript function, however, is different. Looking over this function, you can see that it is expecting a long string of multiple values:

```
function GetCustDetailsFromServer(result, context){
   var i = result.split("|");
   customerID.innerHTML = i[0];
   companyName.innerHTML = i[1];
   contactName.innerHTML = i[2];
   contactTitle.innerHTML = i[3];
   address.innerHTML = i[4];
   city.innerHTML = i[5];
   region.innerHTML = i[6];
   postalCode.innerHTML = i[7];
   country.innerHTML = i[8];
   phone.innerHTML = i[9];
   fax.innerHTML = i[10];
}
```

The multiple results expected are constructed in a pipe-delimited string, and each of the values is placed into an array. Then each string item in the array is assigned to a particular `<span>` tag in the ASP.NET page. For instance, take a look at the following bit of code:

```
customerID.innerHTML = i[0];
```

The `i[0]` variable is the first item found in the pipe-delimited string, and it is assigned to the `customerID` item on the page. This `customerID` identifier comes from the following `<span>` tag found in the table:

```
<span id="customerID" />
```

Now, turn your attention to the code-behind file for this page, as shown in Listing 2-16.

Listing 2-16: The code-behind file for the Customer Details page

VB

```
Imports System.Data
Imports System.Data.SqlClient

Partial Class _Default
    Inherits System.Web.UI.Page
    Implements System.Web.UI.ICallbackEventHandler

    Dim _callbackResult As String = Nothing

    Public Function GetCallbackResult() As String _
       Implements System.Web.UI.ICallbackEventHandler.GetCallbackResult
        Return _callbackResult
    End Function

    Public Sub RaiseCallbackEvent(ByVal eventArgument As String) _
       Implements System.Web.UI.ICallbackEventHandler.RaiseCallbackEvent
```

```
            Dim conn As SqlConnection = New _
               SqlConnection("Data Source=.;Initial Catalog=Northwind;User ID=sa")
            Dim cmd As SqlCommand = New _
               SqlCommand("Select * From Customers Where CustomerID = '" & _
               eventArgument & "'", conn)

            conn.Open()

            Dim MyReader As SqlDataReader
            MyReader = cmd.ExecuteReader(CommandBehavior.CloseConnection)

            Dim MyValues(10) As String

            While MyReader.Read()
                MyValues(0) = MyReader("CustomerID").ToString()
                MyValues(1) = MyReader("CompanyName").ToString()
                MyValues(2) = MyReader("ContactName").ToString()
                MyValues(3) = MyReader("ContactTitle").ToString()
                MyValues(4) = MyReader("Address").ToString()
                MyValues(5) = MyReader("City").ToString()
                MyValues(6) = MyReader("Region").ToString()
                MyValues(7) = MyReader("PostalCode").ToString()
                MyValues(8) = MyReader("Country").ToString()
                MyValues(9) = MyReader("Phone").ToString()
                MyValues(10) = MyReader("Fax").ToString()
            End While

            Conn.Close()

            _callbackResult = String.Join("|", MyValues)
        End Sub

        Protected Sub Page_Load(ByVal sender As Object, _
           ByVal e As System.EventArgs) Handles Me.Load
            Dim cbReference As String = _
               Page.ClientScript.GetCallbackEventReference(Me, "arg", _
                  "GetCustDetailsFromServer", "context")
            Dim cbScript As String = "function UseCallback(arg, context)" & _
               "{" & cbReference & ";" & "}"

            Page.ClientScript.RegisterClientScriptBlock(Me.GetType(), _
               "UseCallback", cbScript, True)
        End Sub
    End Class
```

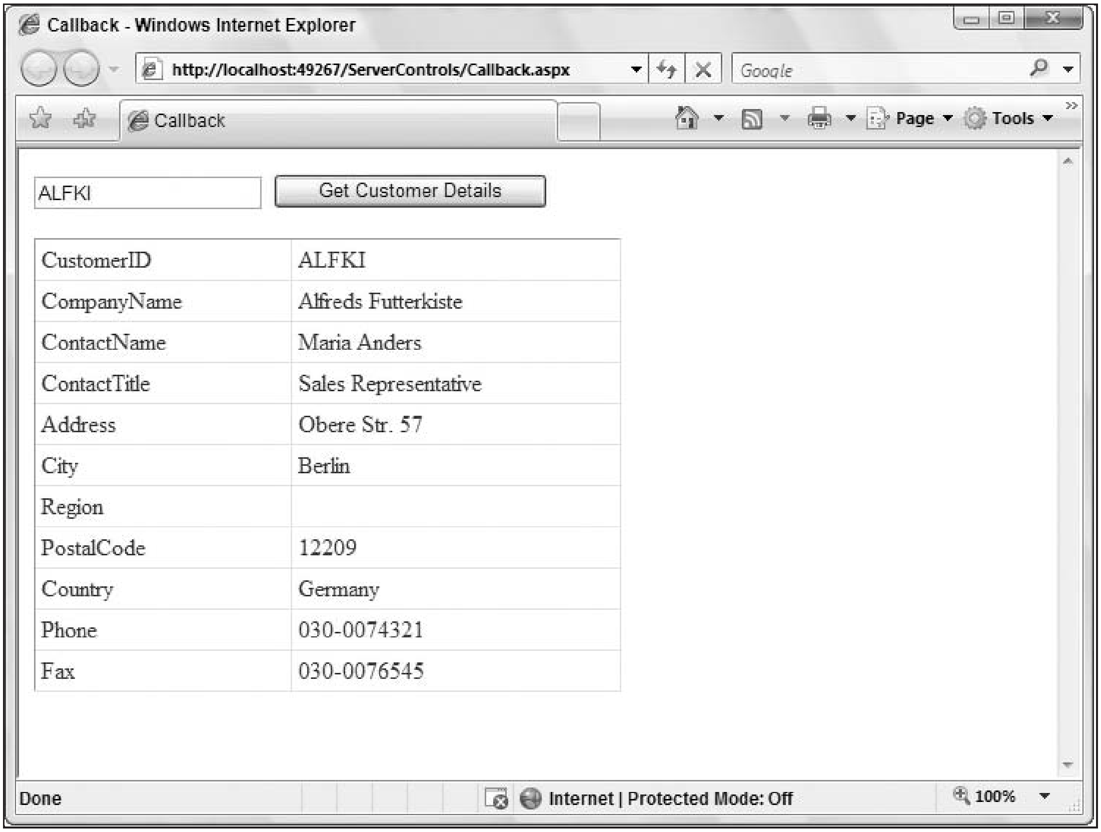
C#

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Data.SqlClient;

public partial class Default2 : System.Web.UI.Page,
   System.Web.UI.ICallbackEventHandler
{
    private string _callbackResult = null;

    protected void Page_Load(object sender, EventArgs e)
    {
        string cbReference = Page.ClientScript.GetCallbackEventReference(this,
           "arg", "GetCustDetailsFromServer", "context");
        string cbScript = "function UseCallback(arg, context)" +
           "{" + cbReference + ";" + "}";

        Page.ClientScript.RegisterClientScriptBlock(this.GetType(),
           "UseCallback", cbScript, true);
    }

    #region ICallbackEventHandler Members

    public string GetCallbackResult()
    {
        return _callbackResult;
    }

    public void RaiseCallbackEvent(string eventArgument)
    {
        SqlConnection conn = new
           SqlConnection("Data Source=.;Initial Catalog=Northwind;User ID=sa");
        SqlCommand cmd = new
           SqlCommand("Select * From Customers Where CustomerID = '" +
           eventArgument + "'", conn);

        conn.Open();

        SqlDataReader MyReader;
        MyReader = cmd.ExecuteReader(CommandBehavior.CloseConnection);

        string[] MyValues = new string[11];

        while (MyReader.Read())
        {
            MyValues[0] = MyReader["CustomerID"].ToString();
            MyValues[1] = MyReader["CompanyName"].ToString();
            MyValues[2] = MyReader["ContactName"].ToString();
            MyValues[3] = MyReader["ContactTitle"].ToString();
            MyValues[4] = MyReader["Address"].ToString();
            MyValues[5] = MyReader["City"].ToString();
            MyValues[6] = MyReader["Region"].ToString();
            MyValues[7] = MyReader["PostalCode"].ToString();
            MyValues[8] = MyReader["Country"].ToString();
            MyValues[9] = MyReader["Phone"].ToString();
```

```
                MyValues[10] = MyReader["Fax"].ToString();
            }

            _callbackResult = String.Join("|", MyValues);
        }

        #endregion
    }
```

Much of this document is quite similar to the document in the previous example using the callback feature. The big difference comes in the `RaiseCallbackEvent()` method. This method first performs a `SELECT` statement on the Customers database based upon the `CustomerID` passed in via the `eventArgument` variable. The result retrieved from this `SELECT` statement is then made part of a string array, which is finally concatenated using the `String.Join()` method before being passed back as the value of the `_callbackResult` object.

With this code in place, you can now populate an entire table of data using the callback feature. This means that the table is populated with no need to refresh the page. The results from this code operation are presented in Figure 2-18.


Callback - Windows Internet Explorer
http://localhost:49267/ServerControls/Callback.aspx
Google
Callback
Page
Tools
Get Customer Details
Done
Internet | Protected Mode: Off
100%

ALFKI

CustomerID	ALFKI
CompanyName	Alfreds Futterkiste
ContactName	Maria Anders
ContactTitle	Sales Representative
Address	Obere Str. 57
City	Berlin
Region	
PostalCode	12209
Country	Germany
Phone	030-0074321
Fax	030-0076545

Figure 2-18

Summary

This chapter gave you one of the core building blocks of an ASP.NET page — the server control. The server control is an object-oriented approach to page development that encapsulates page elements into modifiable and expandable components.

The chapter also introduced you to how to customize the look-and-feel of your server controls using Cascading Style Sheets (CSS). Working with CSS in ASP.NET 3.5 is easy and quick, especially if you have Visual Studio 2008 to assist you. Finally, this chapter looked at both using HTML server controls and adding JavaScript to your pages to modify the behaviors of your controls.

3

ASP.NET Web Server Controls

Of the two types of server controls, HTML server controls and Web server controls, the latter is considered the more powerful and flexible. The previous chapter looked at how to use HTML server controls in applications. HTML server controls enable you to manipulate HTML elements from your server-side code. On the other hand, Web server controls are powerful because they are not explicitly tied to specific HTML elements; rather, they are more closely aligned to the specific functionality that you want to generate. As you will see throughout this chapter, Web server controls can be very simple or rather complex depending on the control you are working with.

The purpose of the large collection of controls is to make you more productive. These controls give you advanced functionality that, in the past, you would have had to laboriously program or simply omit. In the classic ASP days, for example, few calendars were used on Internet Web sites. With the introduction of the Calendar server control in ASP.NET 1.0, calendar creation on a site became a trivial task. Building an image map on top of an image was another task that was difficult to achieve in ASP.NET 1.*x*, but this capability was introduced as a new server control in ASP.NET 2.0.

This chapter introduces some of the available Web server controls. The first part of the chapter focuses on the Web server controls that were around during the ASP.NET 1.0/1.1 days. Then the chapter explores the server controls that were introduced back in ASP.NET 2.0. This chapter does not discuss every possible control because some server controls are introduced and covered in other chapters throughout the book.

An Overview of Web Server Controls

The Web server control is ASP.NET's most-used component. Although you may have been pretty excited by the HTML server controls shown in the previous chapter, Web server controls are definitely a notch higher in capability. They allow for a higher level of functionality that becomes more apparent as you work with them.

The HTML server controls provided by ASP.NET work in that they map to specific HTML elements. You control the output by working with the HTML attributes that the HTML element provides. The attributes can be changed dynamically on the server side before they are finally output to the client. There is a lot of power in this, and you have some HTML server control capabilities that you simply do not have when you work with Web server controls.

Web server controls work differently. They do not map to specific HTML elements, but instead enable you to define functionality, capability, and appearance without the attributes that are available to you through a collection of HTML elements. When constructing a Web page that is made up of Web server controls, you are describing the functionality, the look-and-feel, and the behavior of your page elements. You then let ASP.NET decide how to output this construction. The output, of course, is based on the capabilities of the container that is making the request. This means that each requestor might see a different HTML output because each is requesting the same page with a different browser type or version. ASP.NET takes care of all the browser detection and the work associated with it on your behalf.

Unlike HTML server controls, Web server controls are not only available for working with common Web page form elements (such as text boxes and buttons), but they can also bring some advanced capabilities and functionality to your Web pages. For instance, one common feature of many Web applications is a calendar. No HTML form element places a calendar on your Web forms, but a Web server control from ASP.NET can provide your application with a full-fledged calendar, including some advanced capabilities. In the past, adding calendars to your Web pages was not a small programming task. Today, adding calendars with ASP.NET is rather simple and is achieved with a single line of code!

Remember that when you are constructing your Web server controls, you are actually constructing a control — *a set of instructions* — that is meant for the server (not the client). By default, all Web server controls provided by ASP.NET use an `asp:` at the beginning of the control declaration. The following is a typical Web server control:

```
<asp:Label ID="Label1" runat="server" Text="Hello World"></asp:Label>
```

Like HTML server controls, Web server controls require an `ID` attribute to reference the control in the server-side code, as well as a `runat="server"` attribute declaration. As you do for other XML-based elements, you need to properly open and close Web server controls using XML syntax rules. In the preceding example, you can see the `<asp:Label>` control has a closing `</asp:Label>` element associated with it. You could have also closed this element using the following syntax:

```
<asp:Label ID="Label1" Runat="server" Text="Hello World" />
```

The rest of this chapter examines some of the Web server controls available to you in ASP.NET.

The Label Server Control

The Label server control is used to display text in the browser. Because this is a server control, you can dynamically alter the text from your server-side code. As you saw from the preceding examples of using the `<asp:Label>` control, the control uses the `Text` attribute to assign the content of the control as shown here:

```
<asp:Label ID="Label1" runat="server" Text="Hello World" />
```

Instead of using the `Text` attribute, however, you can place the content to be displayed between the `<asp:Label>` elements like this:

```
<asp:Label ID="Label1" runat="server">Hello World</asp:Label>
```

You can also provide content for the control through programmatic means, as illustrated in Listing 3-1.

Listing 3-1: Programmatically providing text to the Label control

VB

```
Label1.Text = "Hello ASP.NET"
```

C#

```
Label1.Text = "Hello ASP.NET";
```

The Label server control has always been a control that simply showed text. Ever since ASP.NET 2.0, it has a little bit of extra functionality. The big change since this release of the framework is that you can now give items in your form hot-key functionality (also known as *accelerator* keys). This causes the page to focus on a particular server control that you declaratively assign to a specific hot-key press (for example, using Alt+N to focus on the first text box on the form).

A hot key is a quick way for the end user to initiate an action on the page. For instance, if you use Microsoft Internet Explorer, you can press Ctrl+N to open a new instance of IE. Hot keys have always been quite common in thick-client applications (Windows Forms), and now you can use them in ASP.NET. Listing 3-2 shows an example of how to give hot-key functionality to two text boxes on a form.

Listing 3-2: Using the Label server control to provide hot-key functionality

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Label Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <p>
            <asp:Label ID="Label1" runat="server" AccessKey="N"
             AssociatedControlID="Textbox1">User<u>n</u>ame</asp:Label>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox></p>
        <p>
            <asp:Label ID="Label2" runat="server" AccessKey="P"
             AssociatedControlID="Textbox2"><u>P</u>assword</asp:Label>
            <asp:TextBox ID="TextBox2" Runat="server"></asp:TextBox></p>
        <p>
            <asp:Button ID="Button1" runat="server" Text="Submit" />
        </p>
    </form>
</body>
</html>
```

Hot keys are assigned with the `AccessKey` attribute. In this case, `Label1` uses `N`, and `Label2` uses `P`. The second new attribute for the Label control is the `AssociatedControlID` attribute. The `String` value placed here associates the Label control with another server control on the form. The value must be one of the other server controls on the form. If not, the page gives you an error when invoked.

With these two controls in place, when the page is called in the browser, you can press Alt+N or Alt+P to automatically focus on a particular text box in the form. In Figure 3-1, HTML-declared underlines indicate the letters to be pressed along with the Alt key to create focus on the control adjoining the text. This is not required, but we highly recommend it because it is what the end user expects when working with hot keys. In this example, the letter `n` in `Username` and the letter `P` in `Password` are underlined.

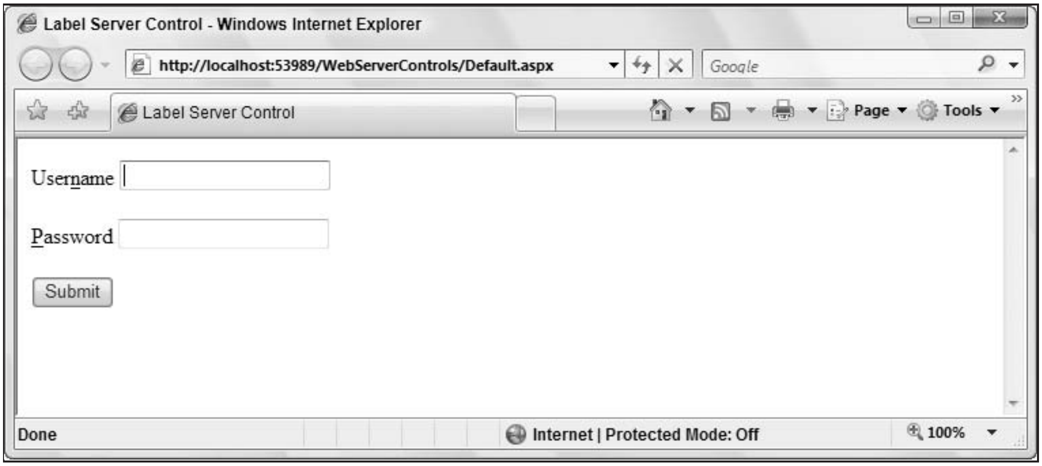

Label Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Label Server Control
Page
Tools
Username
Password
Submit
Done
Internet | Protected Mode: Off
100%

Figure 3-1

When working with hot keys, be aware that not all letters are available to use with the Alt key. Microsoft Internet Explorer already uses Alt+F, E, V, I, O, T, A, W, and H. If you use any of these letters, IE actions supersede any actions you place on the page.

The Literal Server Control

The Literal server control works very much like the Label server control does. This control was always used in the past for text that you wanted to push out to the browser, but keep unchanged in the process (a literal state). A Label control alters the output by placing `<span>` elements around the text as shown:

```
<span id="Label1">Here is some text</span>
```

The Literal control just outputs the text without the `<span>` elements. One feature found in this server control is the attribute `Mode`. This attribute enables you to dictate how the text assigned to the control is interpreted by the ASP.NET engine.

If you place some HTML code in the string that is output (for instance, `<b>Here is some text</b>`), the Literal control outputs just that and the consuming browser shows the text as bold:

```
Here is some text
```

Try using the `Mode` attribute as illustrated here:

```
<asp:Literal ID="Literal1" runat="server" Mode="Encode"
 Text="<b>Here is some text</b>"></asp:Literal>
```

Adding `Mode="Encode"` encodes the output before it is received by the consuming application:

```
&lt;b&gt;Label&lt;/b&gt;
```

Now, instead of the text being converted to a bold font, the `<b>` elements are displayed:

```
<b>Here is some text</b>
```

This is ideal if you want to display code in your application. Other values for the `Mode` attribute include `Transform` and `PassThrough`. `Transform` looks at the consumer and includes or removes elements as needed. For instance, not all devices accept HTML elements so, if the value of the `Mode` attribute is set to `Transform`, these elements are removed from the string before it is sent to the consuming application. A value of `PassThrough` for the `Mode` property means that the text is sent to the consuming application without any changes being made to the string.

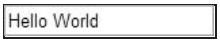
The TextBox Server Control

One of the main features of Web pages is to offer forms that end users can use to submit their information for collection. The TextBox server control is one of the most used controls in this space. As its name suggests, the control provides a text box on the form that enables the end user to input text. You can map the TextBox control to three different HTML elements used in your forms.

First, the TextBox control can be used as a standard HTML text box, as shown in the following code snippet:

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
```

This code creates a text box on the form that looks like the one shown in Figure 3-2.



Hello World

Figure 3-2

Second, the TextBox control can allow end users to input their passwords into a form. This is done by changing the `TextMode` attribute of the TextBox control to `Password`, as illustrated here:

```
<asp:TextBox ID="TextBox1" runat="server" TextMode="Password"></asp:TextBox>
```

When asking end users for their passwords through the browser, it is best practice to provide a text box that encodes the content placed in this form element. Using an attribute and value of `TextMode= "Password"` ensures that the text is encoded with either a star (`*`) or a dot, as shown in Figure 3-3.

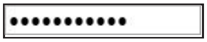
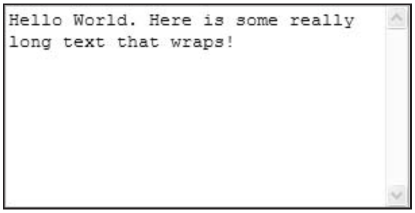


Figure 3-3

Third, the TextBox server control can be used as a multiline text box. The code for accomplishing this task is as follows:

```
<asp:TextBox ID="TextBox1" runat="server" TextMode="MultiLine"
 Width="300px" Height="150px"></asp:TextBox>
```

Giving the `TextMode` attribute a value of `MultiLine` creates a multilined text box in which the end user can enter a larger amount of text in the form. The `Width` and `Height` attributes set the size of the text area, but these are optional attributes — without them, the text area is produced in its smallest size. Figure 3-4 shows the use of the preceding code after adding some text.



```
Hello World. Here is some really
long text that wraps!
```

Figure 3-4

When working with a multilined text box, be aware of the `Wrap` attribute. When set to `True` (which is the default), the text entered into the text area wraps to the next line if needed. When set to `False`, the end user can type continuously in a single line until she presses the Enter key, which brings the cursor down to the next line.

Using the Focus() Method

Because the TextBox server control is derived from the base class of `WebControl`, one of the methods available to it is `Focus()`. The `Focus()` method enables you to dynamically place the end user's cursor in an appointed form element (not just the TextBox control, but in any of the server controls derived from the `WebControl` class). With that said, it is probably most often used with the TextBox control, as illustrated in Listing 3-3.

Listing 3-3: Using the Focus() method with the TextBox control

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   TextBox1.Focus()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
   TextBox1.Focus();
}
```

When the page using this method is loaded in the browser, the cursor is already placed inside of the text box, ready for you to start typing. There is no need to move your mouse to get the cursor in place so you can start entering information in the form. This is ideal for those folks who take a keyboard approach to working with forms.

Using AutoPostBack

ASP.NET pages work in an event-driven way. When an action on a Web page triggers an event, server-side code is initiated. One of the more common events is an end user clicking a button on the form. If you double-click the button in Design view of Visual Studio 2008, you can see the code page with the structure of the `Button1_Click` event already in place. This is because `OnClick` is the most common event of the Button control. Double-clicking the TextBox control constructs an `OnTextChanged` event. This event is triggered when the end user moves the cursor focus outside the text box, either by clicking another element on the page after entering something into a text box, or by simply tabbing out of the text box. The use of this event is shown in Listing 3-4.

Listing 3-4: Triggering an event when a TextBox change occurs

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub TextBox1_TextChanged(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Response.Write("OnTextChanged event triggered")
    End Sub

    Protected Sub Button1_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Response.Write("OnClick event triggered")
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>OnTextChanged Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" runat="server" AutoPostBack="True"
         OnTextChanged="TextBox1_TextChanged"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" Text="Button"
         OnClick="Button1_Click" />
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void TextBox1_TextChanged(object sender, EventArgs e)
```

Continued

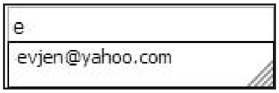
```
        {
            Response.Write("OnTextChanged event triggered");
        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            Response.Write("OnClick event triggered");
        }
    </script>
```

As you build and run this page, notice that you can type something in the text box, but once you tab out of it, the `OnTextChanged` event is triggered and the code contained in the `TextBox1_TextChanged` event runs. To make this work, you must add the `AutoPostBack` attribute to the TextBox control and set it to `True`. This causes the Web page to look for any text changes prior to an actual page postback. For the `AutoPostBack` feature to work, the browser viewing the page must support ECMAScript.

Using AutoCompleteType

You want the forms you build for your Web applications to be as simple to use as possible. You want to make them easy and quick for the end user to fill out the information and proceed. If you make a form too onerous, the people who come to your site may leave without completing it.

One of the great capabilities for any Web form is smart auto-completion. You may have seen this yourself when you visited a site for the first time. As you start to fill out information in a form, a drop-down list appears below the text box as you type, showing you a value that you have typed in a previous form. The plain text box you were working with has become a smart text box. Figure 3-5 shows an example of this feature.



e

evjen@yahoo.com

Figure 3-5

A great new aspect of the TextBox control is the `AutoCompleteType` attribute, which enables you to apply the auto-completion feature to your own forms. You have to help the text boxes on your form to recognize the type of information that they should be looking for. What does that mean? Well, first look at the possible values of the `AutoCompleteType` attribute:

```
BusinessCity
BusinessCountryRegion
BusinessFax
BusinessPhone
BusinessState
BusinessStateAddress
BusinessUrl
BusinessZipCode
Cellular
Company
Department
Disabled
DisplayName
Email
FirstName
Gender
HomeCity
HomeCountryRegion
HomeFax
Homepage
HomePhone
HomeState
HomeStreetAddress
HomeZipCode
JobTitle
LastName
MiddleName
None
Notes
Office
Pager
Search
```

From this list, you can see that if your text box is asking for the end user's home street address, you want to use the following in your TextBox control:

```
<asp:TextBox ID="TextBox1" runat="server"
 AutoCompleteType="HomeStreetAddress"></asp:TextBox>
```

As you view the source of the text box you created, you can see that the following construction has occurred:

```
<input name="TextBox1" type="text" vcard_name="vCard.Home.StreetAddress"
 id="TextBox1" />
```

This feature makes your forms easier to work with. Yes, it is a simple thing, but sometimes it is the little things that keep the viewers coming back again and again to your Web site.

The Button Server Control

Another common control for your Web forms is a button that can be constructed using the Button server control. Buttons are the usual element used to submit forms. Most of the time you are simply dealing with items contained in your forms through the Button control's `OnClick` event, as illustrated in Listing 3-5.

Listing 3-5: The Button control's OnClick event

VB

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
   ' Code here
End Sub
```

C#

```
protected void Button1_Click(object sender, EventArgs e)
{
   // Code here
}
```

The Button control is one of the easier controls to use, but there are a couple of properties of which you must be aware: `CausesValidation` and `CommandName`. They are discussed in the following sections.

The CausesValidation Property

If you have more than one button on your Web page and you are working with the validation server controls, you may not want to fire the validation for each button on the form. Setting the `CausesValidation` property to `False` is a way to use a button that will not fire the validation process. This is explained in more detail in Chapter 4.

The CommandName Property

You can have multiple buttons on your form all working from a single event. The nice thing is that you can also tag the buttons so that the code can make logical decisions based on which button on the form was clicked. You must construct your Button controls in the manner illustrated in Listing 3-6 to take advantage of this behavior.

Listing 3-6: Constructing multiple Button controls to work from a single function

```
<asp:Button ID="Button1" runat="server" Text="Button 1"
 OnCommand="Button_Command" CommandName="DoSomething1" />
<asp:Button ID="Button2" runat="server" Text="Button 2"
 OnCommand="Button_Command" CommandName="DoSomething2" />
```

Looking at these two instances of the Button control, you should pay attention to several things. The first thing to notice is what is not present — any attribute mention of an `OnClick` event. Instead, you use the `OnCommand` event, which points to an event called `Button_Command`. You can see that both Button controls are working from the same event. How does the event differentiate between the two buttons being clicked? Through the value placed in the `CommandName` property. In this case, they are indeed separate values — `DoSomething1` and `DoSomething2`.

The next step is to create the `Button_Command` event to deal with both these buttons by simply typing one out or by selecting the `Command` event from the drop-down list of available events for the Button control from the code view of Visual Studio. In either case, you should end up with an event like the one shown in Listing 3-7.

Listing 3-7: The Button_Command event

VB

```
Protected Sub Button_Command(ByVal sender As Object, _
  ByVal e As System.Web.UI.WebControls.CommandEventArgs)

   Select Case e.CommandName
      Case "DoSomething1"
         Response.Write("Button 1 was selected")
      Case "DoSomething2"
         Response.Write("Button 2 was selected")
   End Select

End Sub
```

C#

```
protected void Button_Command(Object sender,
   System.Web.UI.WebControls.CommandEventArgs e)
{
   switch (e.CommandName)
   {
      case("DoSomething1"):
         Response.Write("Button 1 was selected");
         break;
      case("DoSomething2"):
         Response.Write("Button 2 was selected");
         break;
   }
}
```

Notice that this method uses `System.Web.UI.WebControls.CommandEventArgs` instead of the typical `System.EventArgs`. This gives you access to the member `CommandName` used in the `Select Case` (`switch`) statement as `e.CommandName`. Using this object, you can check for the value of the `CommandName` property used by the button that was clicked on the form and take a specific action based upon the value passed.

You can add some parameters to be passed in to the `Command` event beyond what is defined in the `CommandName` property. You do this by using the Button control's `CommandArgument` property. Adding values to the property enables you to define items a bit more granularly if you want. You can get at this value via server-side code using `e.CommandArgument` from the `CommandEventArgs` object.

Buttons That Work with Client-Side JavaScript

Buttons are frequently used for submitting information and causing actions to occur on a Web page. Before ASP.NET 1.0/1.1, people intermingled quite a bit of JavaScript in their pages to fire JavaScript events when a button was clicked. The process became more cumbersome in ASP.NET 1.0/1.1, but ever since ASP.NET 2.0, it is much easier.

You can create a page that has a JavaScript event, as well as a server-side event, triggered when the button is clicked, as illustrated in Listing 3-8.

Listing 3-8: Two types of events for the button

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Response.Write("Postback!")
    End Sub
</script>

<script language="javascript">
   function AlertHello()
   {
      alert('Hello ASP.NET');
   }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Button Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Button ID="Button1" runat="server" Text="Button"
         OnClientClick="AlertHello()" OnClick="Button1_Click" />
    </form>
```

Continued

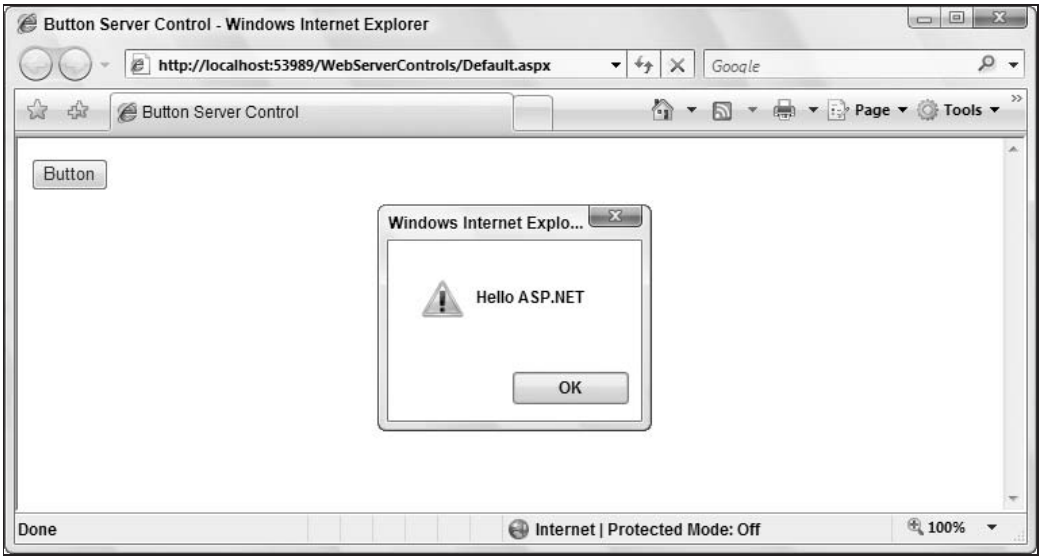
```
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Response.Write("Postback!");
    }
</script>
```

The first thing to notice is the new attribute for the Button server control: `OnClientClick`. It points to the client-side function, unlike the `OnClick` attribute that points to the server-side event. This example uses a JavaScript function called `AlertHello()`.

One cool thing about Visual Studio 2008 is that it can work with server-side script tags that are right alongside client-side script tags. It all works together seamlessly. In the example, after the JavaScript alert dialog is issued (see Figure 3-6) and the end user clicks OK, the page posts back as the server-side event is triggered.


Button Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Button Server Control
Page
Tools
Button
Windows Internet Explo...
OK
Done
Internet | Protected Mode: Off
100%

Hello ASP.NET

Figure 3-6

Another interesting attribute for the button controls is `PostBackUrl`. It enables you to perform cross-page posting, instead of simply posting your ASP.NET pages back to the same page, as shown in the following example:

```
<asp:Button ID="Button2" runat="server" Text="Submit page to Page2.aspx"
 PostBackUrl="Page2.aspx" />
```

Cross-page posting is covered in greater detail in Chapter 1.

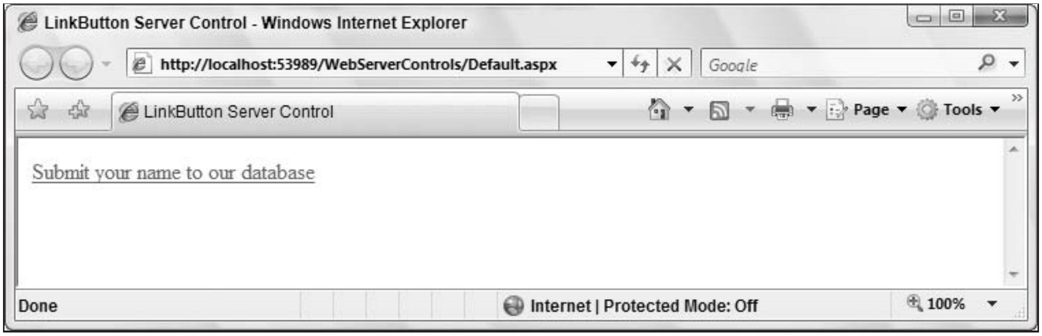
The LinkButton Server Control

The LinkButton server control is a variation of the Button control. It is the same except that the LinkButton control takes the form of a hyperlink. Nevertheless, it is not a typical hyperlink. When the end user clicks the link, it behaves like a button. This is an ideal control to use if you have a large number of buttons on your Web form.

A LinkButton server control is constructed as follows:

```
<asp:LinkButton ID="LinkButton1" Runat="server" OnClick="LinkButton1_Click">
   Submit your name to our database
</asp:LinkButton>
```

Using the LinkButton control gives you the results shown in Figure 3-7.


LinkButton Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
LinkButton Server Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

Submit your name to our database

Figure 3-7

The ImageButton Server Control

The ImageButton control is also a variation of the Button control. It is almost exactly the same as the Button control except that it enables you to use a custom image as the form's button instead of the typical buttons used on most forms. This means that you can create your own buttons as images and the end users can click the images to submit form data. A typical construction of the ImageButton is as follows:

```
<asp:ImageButton ID="ImageButton1" runat="server"
 OnClick="ImageButton1_Click" ImageUrl="MyButton.jpg" />
```

The ImageButton control specifies the location of the image used by using the `ImageUrl` property. From this example, you can see that the `ImageUrl` points to `MyButton.jpg`. The big difference between the ImageButton control and the LinkButton or Button controls is that ImageButton takes a different construction for the `OnClick` event. It is shown in Listing 3-9.

Listing 3-9: The Click event for the ImageButton control

VB

```
Protected Sub ImageButton1_Click(ByVal sender As Object, _
   ByVal e As System.Web.UI.WebControls.ImageClickEventArgs)
```

Continued

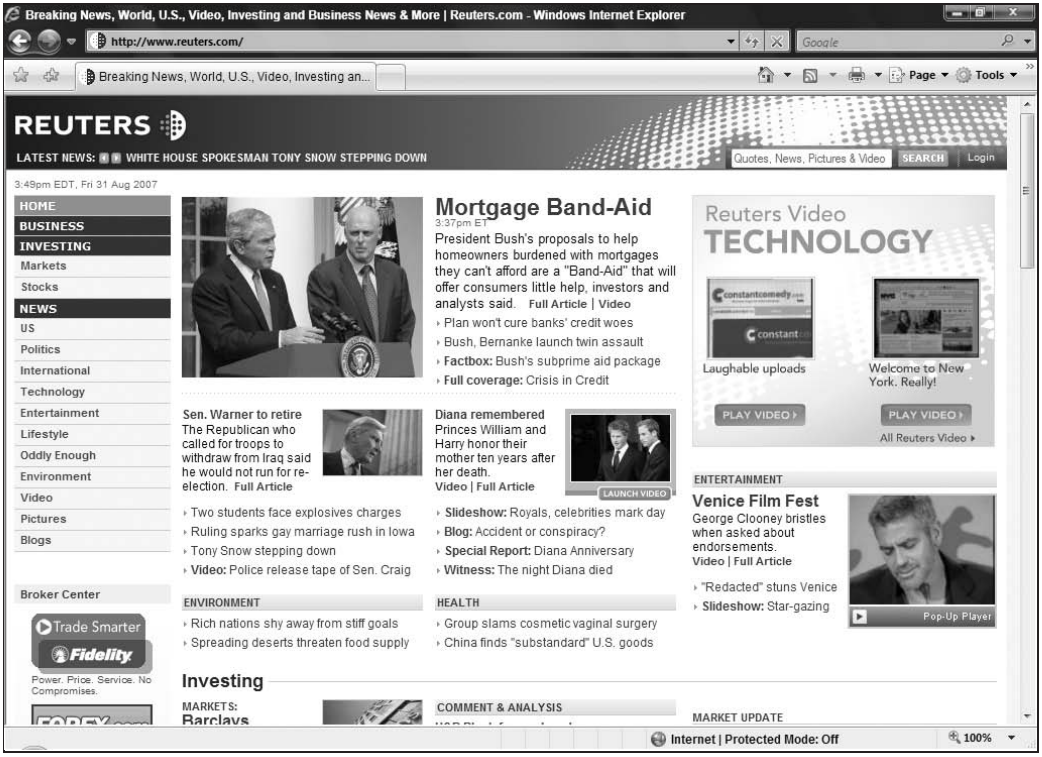
```
   ' Code here
End Sub
```

C#

```
protected void ImageButton1_Click(object sender,
   System.Web.UI.WebControls.ImageClickEventArgs e)
{
   // Code here
}
```

The construction uses the `ImageClickEventArgs` object instead of the `System.EventArgs` object usually used with the LinkButton and Button controls. You can use this object to determine where in the image the end user clicked by using both `e.X` and `e.Y` coordinates.

The Search and Play Video buttons on the page shown in Figure 3-8 are image buttons.


Breaking News, World, U.S., Video, Investing and Business News & More | Reuters.com - Windows Internet Explorer
http://www.reuters.com/
Google
Breaking News, World, U.S., Video, Investing an...
Page
Tools
REUTERS
LATEST NEWS: WHITE HOUSE SPOKESMAN TONY SNOW STEPPING DOWN
Quotes, News, Pictures & Video
SEARCH
Login
3:49pm EDT, Fri 31 Aug 2007
HOME
BUSINESS
INVESTING
Markets
Stocks
NEWS
US
Politics
International
Technology
Entertainment
Lifestyle
Oddly Enough
Environment
Video
Pictures
Blogs
Broker Center
Trade Smarter
Fidelity
Power. Price. Service. No Compromises.
FOREX.com
Reuters Video
TECHNOLOGY
Laughable uploads
Welcome to New York. Really!
PLAY VIDEO
PLAY VIDEO
All Reuters Video
LAUNCH VIDEO
Pop-Up Player
Internet | Protected Mode: Off
100%

Mortgage Band-Aid
3:37pm ET
President Bush's proposals to help homeowners burdened with mortgages they can't afford are a "Band-Aid" that will offer consumers little help, investors and analysts said. Full Article | Video
- Plan won't cure banks' credit woes
- Bush, Bernanke launch twin assault
- **Factbox:** Bush's subprime aid package
- **Full coverage:** Crisis in Credit

Sen. Warner to retire
The Republican who called for troops to withdraw from Iraq said he would not run for re-election. Full Article
- Two students face explosives charges
- Ruling sparks gay marriage rush in Iowa
- Tony Snow stepping down
- **Video:** Police release tape of Sen. Craig

Diana remembered
Princes William and Harry honor their mother ten years after her death.
Video | Full Article
- **Slideshow:** Royals, celebrities mark day
- **Blog:** Accident or conspiracy?
- **Special Report:** Diana Anniversary
- **Witness:** The night Diana died

ENTERTAINMENT
Venice Film Fest
George Clooney bristles when asked about endorsements.
Video | Full Article
- "Redacted" stuns Venice
- **Slideshow:** Star-gazing

ENVIRONMENT
- Rich nations shy away from stiff goals
- Spreading deserts threaten food supply

HEALTH
- Group slams cosmetic vaginal surgery
- China finds "substandard" U.S. goods

Investing
MARKETS:
Barclays

COMMENT & ANALYSIS

MARKET UPDATE

Figure 3-8

The HyperLink Server Control

The HyperLink server control enables you to programmatically work with any hyperlinks on your Web pages. Hyperlinks are links that allow end users to transfer from one page to another. You can set the text of a hyperlink using the control's `Text` attribute:

```
<asp:HyperLink ID="HyperLink1" runat="server" Text="Go to this page here"
 NavigateUrl="~/Default2.aspx"></asp:HyperLink>
```

This server control creates a hyperlink on your page with the text `Go to this page here`. When the link is clicked, the user is redirected to the value that is placed in the `NavigateUrl` property (in this case, the `Default2.aspx` page).

The interesting thing about the HyperLink server control is that it can be used for images as well as text. Instead of using the `Text` attribute, it uses the `ImageUrl` property:

```
<asp:HyperLink ID="HyperLink1" runat="server" ImageUrl="~/MyLinkImage.gif"
 NavigateUrl="~/Default2.aspx"></asp:HyperLink>
```

The HyperLink control is a great way to dynamically place hyperlinks on a Web page based either upon user input in a form or on database values that are retrieved when the page is loaded.

The DropDownList Server Control

The DropDownList server control enables you to place an HTML select box on your Web page and program against it. It is ideal when you have a large collection of items from which you want the end user to select a single item. It is usually used for a medium- to large-sized collection. If the collection size is relatively small, consider using the RadioButtonList server control (described later in this chapter).

The select box generated by the DropDownList control displays a single item and allows the end user to make a selection from a larger list of items. Depending on the number of choices available in the select box, the end user may have to scroll through a list of items. Note that the appearance of the scroll bar in the drop-down list is automatically created by the browser depending on the browser version and the number of items contained in the list.

Here is the code for DropDownList control:

```
<asp:DropDownList ID="DropDownList1" runat="server">
   <asp:ListItem>Car</asp:ListItem>
   <asp:ListItem>Airplane</asp:ListItem>
   <asp:ListItem>Train</asp:ListItem>
</asp:DropDownList>
```

This code generates a drop-down list in the browser, as shown in Figure 3-9.

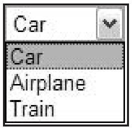


Figure 3-9

The DropDownList control comes in handy when you start binding it to various data stores. The data stores can either be arrays, database values, XML file values, or values found elsewhere. For an example of binding the DropDownList control, this next example looks at dynamically generating a DropDownList control from one of three available arrays, as shown in Listing 3-10.

Listing 3-10: Dynamically generating a DropDownList control from an array

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub DropDownList1_SelectedIndexChanged(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        Dim CarArray() As String = {"Ford", "Honda", "BMW", "Dodge"}
        Dim AirplaneArray() As String = {"Boeing 777", "Boeing 747", "Boeing 737"}
        Dim TrainArray() As String = {"Bullet Train", "Amtrack", "Tram"}

        If DropDownList1.SelectedValue = "Car" Then
            DropDownList2.DataSource = CarArray
        ElseIf DropDownList1.SelectedValue = "Airplane" Then
            DropDownList2.DataSource = AirplaneArray
        Else
            DropDownList2.DataSource = TrainArray
        End If

        DropDownList2.DataBind()
        DropDownList2.Visible = True
    End Sub

    Protected Sub Button1_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Response.Write("You selected <b>" & _
           DropDownList1.SelectedValue.ToString() & ": " & _
           DropDownList2.SelectedValue.ToString() & "</b>")
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>DropDownList Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        Select transportation type:<br />
        <asp:DropDownList ID="DropDownList1" runat="server"
         OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged"
         AutoPostBack="true">
            <asp:ListItem>Select an Item</asp:ListItem>
            <asp:ListItem>Car</asp:ListItem>
            <asp:ListItem>Airplane</asp:ListItem>
            <asp:ListItem>Train</asp:ListItem>
        </asp:DropDownList>&nbsp;
        <asp:DropDownList ID="DropDownList2" runat="server" Visible="false">
        </asp:DropDownList>
        <asp:Button ID="Button1" runat="server" Text="Select Options"
         OnClick="Button1_Click" />
    </div>
    </form>
</body>
</html>
```

C#

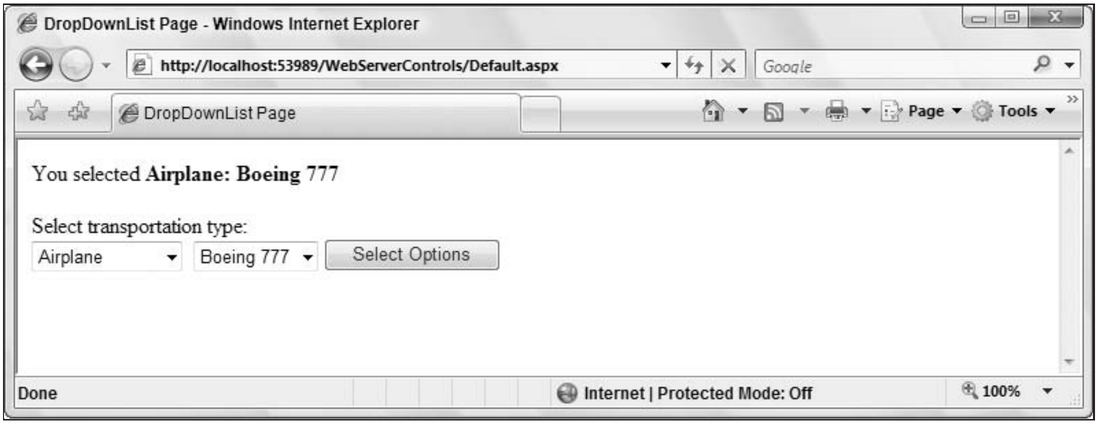
```
<%@ Page Language="C#" %>

<script runat="server">
    protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
    {
        string[] CarArray = new string[4] {"Ford", "Honda", "BMW", "Dodge"};
        string[] AirplaneArray = new string[3] {"Boeing 777", "Boeing 747",
           "Boeing 737"};
        string[] TrainArray = new string[3] {"Bullet Train", "Amtrack", "Tram"};

        if (DropDownList1.SelectedValue == "Car") {
            DropDownList2.DataSource = CarArray; }
        else if (DropDownList1.SelectedValue == "Airplane") {
            DropDownList2.DataSource = AirplaneArray; }
        else {
            DropDownList2.DataSource = TrainArray;
        }

        DropDownList2.DataBind();
        DropDownList2.Visible = true;
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        Response.Write("You selected <b>" +
           DropDownList1.SelectedValue.ToString() + ": " +
           DropDownList2.SelectedValue.ToString() + "</b>");
    }
</script>
```

In this example, the second drop-down list is dynamically generated based upon the value selected from the first drop-down list. For instance, selecting `Car` from the first drop-down list dynamically creates a second drop-down list on the form that includes a list of available car selections.

This is possible because of the use of the `AutoPostBack` feature of the `DropDownList` control. When the `AutoPostBack` property is set to `True`, the method provided through the `OnSelectedIndexChanged` event is fired when a selection is made. In the example, the `DropDownList1_SelectedIndexChanged` event is fired, dynamically creating the second drop-down list.

In this method, the content of the second drop-down list is created in a string array and then bound to the second DropDownList control through the use of the `DataSource` property and the `DataBind()` method.

When built and run, this page looks like the one shown in Figure 3-10.


DropDownList Page - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
DropDownList Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

You selected **Airplane: Boeing 777**

Select transportation type:

Airplane
Boeing 777
Select Options

Figure 3-10

Visually Removing Items from a Collection

The DropDownList, ListBox, CheckBoxList, and RadioButtonList server controls give you the capability to visually remove items from the collection displayed in the control, although you can still work with the items that are not displayed in your server-side code.

The ListBox, CheckBoxList, and RadioButtonList controls are discussed shortly in this chapter.

For a quick example of removing items, create a drop-down list with three items, including one that you will not display. On the postback, however, you can still work with the ListItem's `Value` or `Text` property, as illustrated in Listing 3-11.

Listing 3-11: Disabling certain ListItems from a collection

VB

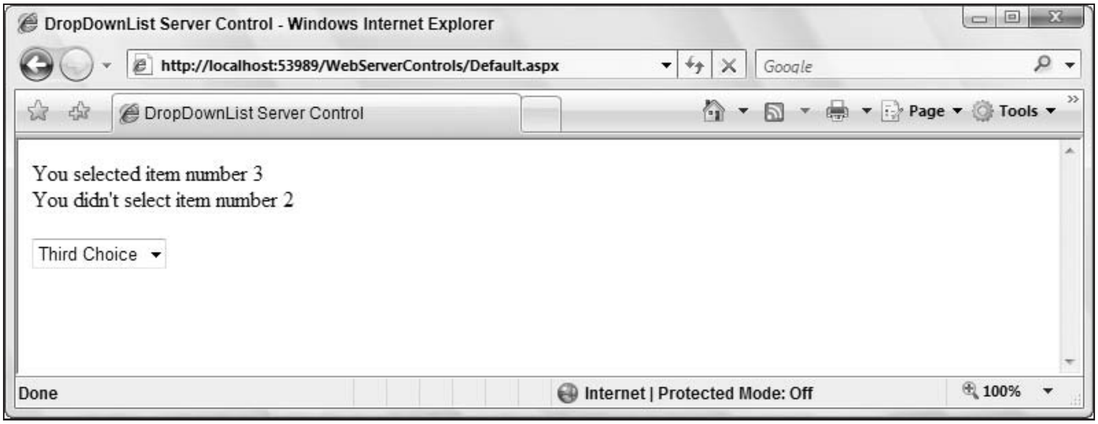
```
<%@ page language="VB" %>

<script runat="server">
    Protected Sub DropDownList1_SelectedIndexChanged(ByVal sender As Object, _
       ByVal e As System.EventArgs)
         Response.Write("You selected item number " & _
           DropDownList1.SelectedValue & "<br>")
         Response.Write("You didn't select item number " & _
           DropDownList1.Items(1).Value)
    End Sub
</script>

<html>
<head runat="server">
    <title>DropDownList Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
```

```
        <asp:DropDownList ID="DropDownList1" Runat="server" AutoPostBack="True"
         OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
            <asp:ListItem Value="1">First Choice</asp:ListItem>
            <asp:ListItem Value="2" Enabled="False">Second Choice</asp:ListItem>
            <asp:ListItem Value="3">Third Choice</asp:ListItem>
        </asp:DropDownList>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
    {
         Response.Write("You selected item number " +
           DropDownList1.SelectedValue + "<br>");
         Response.Write("You didn't select item number " +
           DropDownList1.Items[1].Value);
    }
</script>
```

From the code, you can see that the `<asp:ListItem>` element has an attribute: `Enabled`. The Boolean value given to this element dictates whether an item in the collection is displayed. If you use `Enabled="False"`, the item is not displayed, but you still have the capability to work with the item in the server-side code displayed in the `DropDownList1_SelectedIndexChanged` event. The result of the output from these `Response.Write` statements is shown in Figure 3-11.


DropDownList Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
DropDownList Server Control
Page
Tools

You selected item number 3
You didn't select item number 2

Third Choice
Done
Internet | Protected Mode: Off
100%

Figure 3-11

The ListBox Server Control

The ListBox server control has a function similar to the DropDownList control. It displays a collection of items. The ListBox control behaves differently from the DropDownList control in that it displays more of the collection to the end user, and it enables the end user to make multiple selections from the collection — something that is not possible with the DropDownList control.

A typical ListBox control appears in code as follows:

```
<asp:ListBox ID="ListBox1" runat="server">
   <asp:ListItem>Hematite</asp:ListItem>
   <asp:ListItem>Halite</asp:ListItem>
   <asp:ListItem>Limonite</asp:ListItem>
   <asp:ListItem>Magnetite</asp:ListItem>
</asp:ListBox>
```

This generates the browser display illustrated in Figure 3-12.

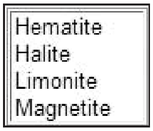


Figure 3-12

Allowing Users to Select Multiple Items

You can use the `SelectionMode` attribute to let your end users make multiple selections from what is displayed by the ListBox control. Here's an example:

```
<asp:ListBox ID="ListBox1" runat="server" SelectionMode="Multiple">
   <asp:ListItem>Hematite</asp:ListItem>
   <asp:ListItem>Halite</asp:ListItem>
   <asp:ListItem>Limonite</asp:ListItem>
   <asp:ListItem>Magnetite</asp:ListItem>
</asp:ListBox>
```

The possible values of the `SelectionMode` property include `Single` and `Multiple`. Setting the value to `Multiple` allows the end user to make multiple selections in the list box. The user must hold down either the Ctrl or Shift keys while making selections. Holding down the Ctrl key enables the user to make a single selection from the list while maintaining previous selections. Holding down the Shift key enables a range of multiple selections.

An Example of Using the ListBox Control

The ListBox control shown in Listing 3-12 allows multiple selections to be displayed in the browser when a user clicks the Submit button. The form should also have an additional text box and button at the top that enables the end user to add additional items to the ListBox.

Listing 3-12: Using the ListBox control

VB

```
<%@ Page Language="VB" %>
```

Continued

```
<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        ListBox1.Items.Add(TextBox1.Text.ToString())
    End Sub

    Protected Sub Button2_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Label1.Text = "You selected from the ListBox:<br>"
        For Each li As ListItem In ListBox1.Items
            If li.Selected = True Then
                label1.Text += li.Text & "<br>"
            End If
        Next
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Using the ListBox</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" Text="Add an additional item"
         OnClick="Button1_Click" /><br /><br />

        <asp:ListBox ID="ListBox1" runat="server" SelectionMode="multiple">
           <asp:ListItem>Hematite</asp:ListItem>
           <asp:ListItem>Halite</asp:ListItem>
           <asp:ListItem>Limonite</asp:ListItem>
           <asp:ListItem>Magnetite</asp:ListItem>
        </asp:ListBox><br /><br />

        <asp:Button ID="Button2" runat="server" Text="Submit"
         OnClick="Button2_Click" /><br /><br />

        <asp:Label ID="Label1" runat="server"></asp:Label>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
```

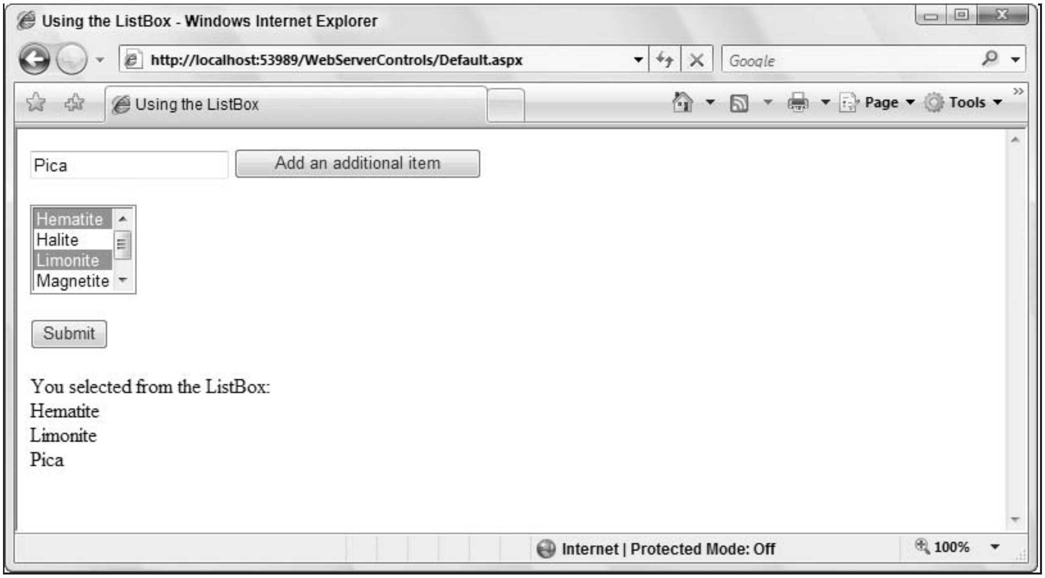
Continued

```
        {
           ListBox1.Items.Add(TextBox1.Text.ToString());
        }

        protected void Button2_Click(object sender, EventArgs e)
        {
           Label1.Text = "You selected from the ListBox:<br>";
           foreach (ListItem li in ListBox1.Items) {
               if (li.Selected) {
                   Label1.Text += li.Text + "<br>";
               }
           }
        }
    </script>
```

This is an interesting example. First, some default items (four common minerals) are already placed inside the ListBox control. However, the text box and button at the top of the form allow the end user to add additional minerals to the list. Users can then make one or more selections from the ListBox, including selections from the items that they dynamically added to the collection. After a user makes his selection and clicks the button, the `Button2_Click` event iterates through the `ListItem` instances in the collection and displays only the items that have been selected.

This control works by creating an instance of a `ListItem` object and using its `Selected` property to see if a particular item in the collection has been selected. The use of the `ListItem` object is not limited to the ListBox control (although that is what is used here). You can dynamically add or remove items from a collection and get at items and their values using the `ListItem` object in the DropDownList, CheckBoxList, and RadioButtonList controls as well. It is a list-control feature.

When this page is built and run, you get the results presented in Figure 3-13.


Using the ListBox - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Using the ListBox
Page
Tools
Pica
Add an additional item
Hematite
Halite
Limonite
Magnetite
Submit
Internet | Protected Mode: Off
100%

You selected from the ListBox:
Hematite
Limonite
Pica

Figure 3-13

Adding Items to a Collection

To add items to the collection, you can use the following short syntax:

```
ListBox1.Items.Add(TextBox1.Text)
```

Look at the source code created in the browser, and you should see something similar to the following generated dynamically:

```
<select size="4" name="ListBox1" multiple="multiple" id="ListBox1">
     <option value="Hematite">Hematite</option>
     <option value="Halite">Halite</option>
     <option value="Limonite">Limonite</option>
     <option value="Magnetite">Magnetite</option>
     <option value="Olivine">Olivine</option>
</select>
```

You can see that the dynamically added value is a text item, and you can see its value. You can also add instances of the ListItem object to get different values for the item name and value:

VB

```
ListBox1.Items.Add(New ListItem("Olivine", "MG2SIO4"))
```

C#

```
ListBox1.Items.Add(new ListItem("Olivine", "MG2SIO4"));
```

This example adds a new instance of the `ListItem` object — adding not only the textual name of the item, but also the value of the item (its chemical formula). It produces the following results in the browser:

```
<option value="MG2SIO4">Olivine</option>
```

The CheckBox Server Control

Check boxes on a Web form enable your users to either make selections from a collection of items or specify a value of an item to be yes/no, on/off, or true/false. Use either the CheckBox control or the CheckBoxList control to include check boxes in your Web forms.

The CheckBox control allows you to place single check boxes on a form; the CheckBoxList control allows you to place collections of check boxes on the form. You can use multiple CheckBox controls on your ASP.NET pages, but then you are treating each check box as its own element with its own associated events. On the other hand, the CheckBoxList control allows you to take multiple check boxes and create specific events for the entire group.

Listing 3-13 shows an example of using the CheckBox control.

Listing 3-13: Using a single instance of the CheckBox control

VB

```
<%@ Page Language="VB" %>
```

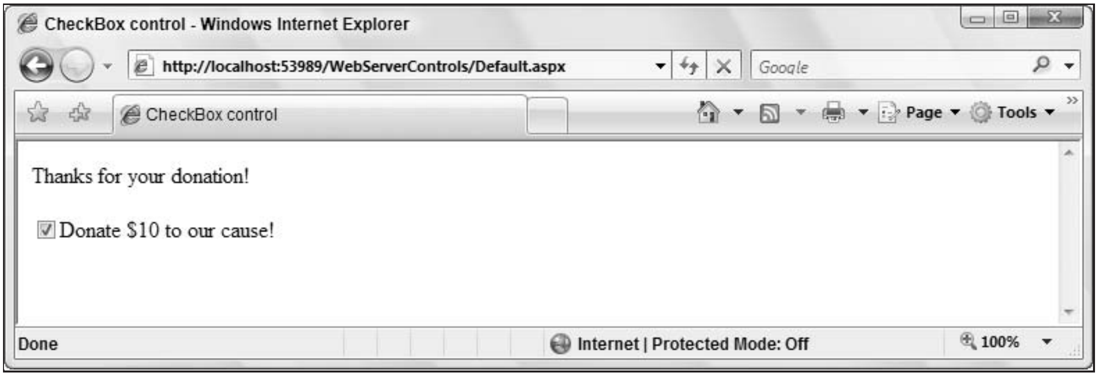
Continued

```
<script runat="server">
    Protected Sub CheckBox1_CheckedChanged(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        Response.Write("Thanks for your donation!")
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CheckBox control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:CheckBox ID="CheckBox1" runat="server" Text="Donate $10 to our cause!"
         OnCheckedChanged="CheckBox1_CheckedChanged" AutoPostBack="true" />
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void CheckBox1_CheckedChanged(object sender, EventArgs e)
    {
        Response.Write("Thanks for your donation!");
    }
</script>
```

This produces a page that contains a single check box asking for a monetary donation. Using the `CheckedChanged` event, `OnCheckedChanged` is used within the CheckBox control. The attribute's value points to the `CheckBox1_CheckedChanged` event, which fires when the user checks the check box. It occurs only if the `AutoPostBack` property is set to `True` (this property is set to `False` by default). Running this page produces the results shown in Figure 3-14.


CheckBox control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
CheckBox control
Page
Tools
Done
Internet | Protected Mode: Off
100%

Thanks for your donation!

☑ Donate $10 to our cause!

Figure 3-14

How to Determine Whether Check Boxes Are Checked

You might not want to use the `AutoPostBack` feature of the check box, but instead want to determine if the check box is checked after the form is posted back to the server. You can make this check through an `If Then` statement, as illustrated in the following example:

VB

```
If (CheckBox1.Checked = True) Then
   Response.Write("CheckBox is checked!")
End If
```

C#

```
if (CheckBox1.Checked == true) {
   Response.Write("Checkbox is checked!");
}
```

This check is done on the CheckBox value using the control's `Checked` property. The property's value is a Boolean value, so it is either `True` (checked) or `False` (not checked).

Assigning a Value to a Check Box

You can also use the `Checked` property to make sure a check box is checked based on other dynamic values:

VB

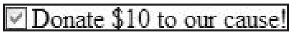
```
If (Member = True) Then
   CheckBox1.Checked = True
End If
```

C#

```
if (Member == true) {
   CheckBox1.Checked = true;
}
```

Aligning Text Around the Check Box

In the previous check box example, the text appears to the right of the actual check box, as shown in Figure 3-15.



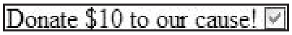
☑ Donate $10 to our cause!

Figure 3-15

Using the CheckBox control's `TextAlign` property, you can realign the text so that it appears on the other side of the check box:

```
<asp:CheckBox ID="CheckBox1" runat="server" Text="Donate $10 to our cause!"
 OnCheckedChanged="CheckBox1_CheckedChanged" AutoPostBack="true"
 TextAlign="Left" />
```

The possible values of the `TextAlign` property are either `Right` (the default setting) or `Left`. This property is also available to the CheckBoxList, RadioButton, and RadioButtonList controls. Assigning the value `Left` produces the result shown in Figure 3-16.



Donate $10 to our cause! ☑

Figure 3-16

The CheckBoxList Server Control

The CheckBoxList server control is quite similar to the CheckBox control, except that the former enables you to work with a collection of items rather than a single item. The idea is that a CheckBoxList server control instance is a collection of related items, each being a check box unto itself.

To see the CheckBoxList control in action, you can build an example that uses Microsoft's SQL Server to pull information from the Customers table of the Northwind example database. An example is presented in Listing 3-14.

Listing 3-14: Dynamically populating a CheckBoxList

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Label1.Text = "You selected:<br>"
        For Each li As ListItem In CheckBoxList1.Items
            If li.Selected = True Then
                Label1.Text += li.Text & "<br>"
            End If
        Next
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CheckBoxList control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Button ID="Button1" runat="server" Text="Submit Choices"
         OnClick="Button1_Click" /><br /><br />
        <asp:Label ID="Label1" runat="server"></asp:Label>
        <br />
        <asp:CheckBoxList ID="CheckBoxList1" runat="server"
         DataSourceID="SqlDataSource1" DataTextField="CompanyName"
         RepeatColumns="3" BorderColor="Black"
         BorderStyle="Solid" BorderWidth="1px">
        </asp:CheckBoxList>
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
         SelectCommand="SELECT [CompanyName] FROM [Customers]"
            ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>">
        </asp:SqlDataSource>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
       Label1.Text = "You selected:<br>";
       foreach (ListItem li in CheckBoxList1.Items) {
           if (li.Selected == true) {
               Label1.Text += li.Text + "<br>";
           }
       }
    }
</script>
```

This ASP.NET page has a SqlDataSource control on the page that pulls the information you need from the Northwind database. From the `SELECT` statement used in this control, you can see that you are retrieving the `CompanyName` field from each of the listings in the Customers table.

The CheckBoxList control binds itself to the SqlDataSource control using a few properties:

```
<asp:CheckBoxList ID="CheckBoxList1" runat="server"
 DataSourceID="SqlDataSource1" DataTextField="CompanyName"
 RepeatColumns="3" BorderColor="Black"
 BorderStyle="Solid" BorderWidth="1px">
</asp:CheckBoxList>
```

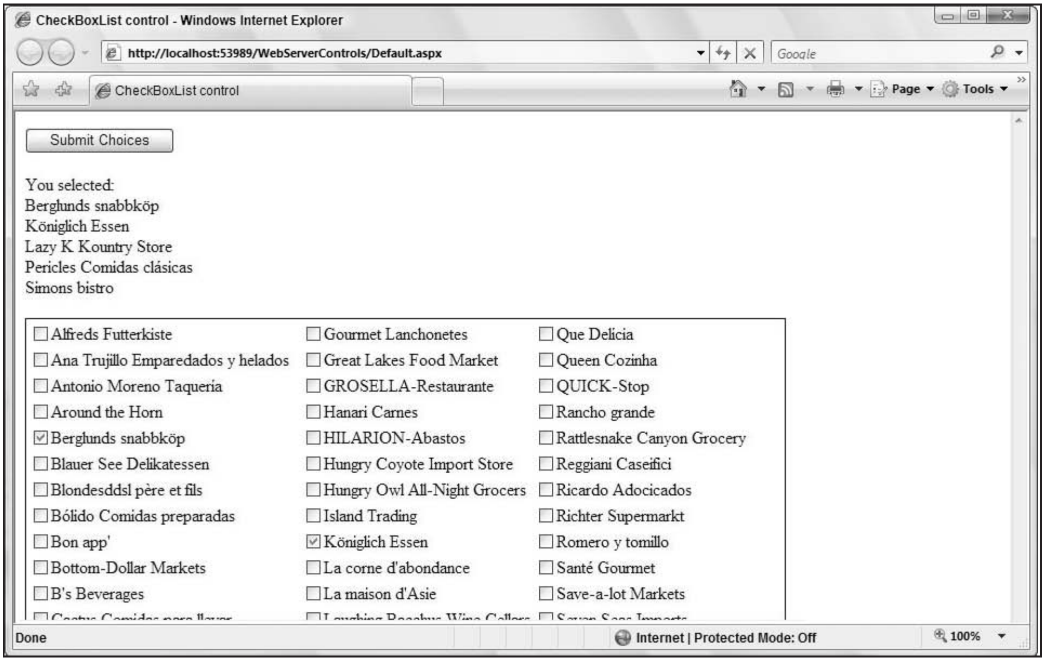
The `DataSourceID` property is used to associate the CheckBoxList control with the results that come back from the SqlDataSource control. Then the `DataTextField` property is used to retrieve the name of the field you want to work with from the results. In this example, it is the only one that is available: the `CompanyName`. That's it! CheckBoxList generates the results you want.

The remaining code consists of styling properties, which.are pretty interesting. The `BorderColor`, `BorderStyle`, and `BorderWidth` properties enable you to put a border around the entire check box list. The most interesting property is the `RepeatColumns` property, which specifies how many columns (three in this example) can be used to display the results.

When you run the page, you get the results shown in Figure 3-17.

The `RepeatDirection` property instructs the CheckBoxList control about how to lay out the items bound to the control on the Web page. Possible values include `Vertical` and `Horizontal`. The default value is `Vertical`. Setting it to `Vertical` with a `RepeatColumn` setting of `3` gives the following results:

```
CheckBox1    CheckBox5    CheckBox9
CheckBox2    CheckBox6    CheckBox10
CheckBox3    CheckBox7    CheckBox11
CheckBox4    CheckBox8    CheckBox12
```


CheckBoxList control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
CheckBoxList control
Page
Tools
Submit Choices
Done
Internet | Protected Mode: Off
100%

You selected:
Berglunds snabbköp
Königlich Essen
Lazy K Kountry Store
Pericles Comidas clásicas
Simons bistro

☐ Alfreds Futterkiste
☐ Ana Trujillo Emparedados y helados
☐ Antonio Moreno Taquería
☐ Around the Horn
☑ Berglunds snabbköp
☐ Blauer See Delikatessen
☐ Blondesddsl père et fils
☐ Bólido Comidas preparadas
☐ Bon app'
☐ Bottom-Dollar Markets
☐ B's Beverages
☐ Gourmet Lanchonetes
☐ Great Lakes Food Market
☐ GROSELLA-Restaurante
☐ Hanari Carnes
☐ HILARION-Abastos
☐ Hungry Coyote Import Store
☐ Hungry Owl All-Night Grocers
☐ Island Trading
☑ Königlich Essen
☐ La corne d'abondance
☐ La maison d'Asie
☐ Que Delícia
☐ Queen Cozinha
☐ QUICK-Stop
☐ Rancho grande
☐ Rattlesnake Canyon Grocery
☐ Reggiani Caseifici
☐ Ricardo Adocicados
☐ Richter Supermarkt
☐ Romero y tomillo
☐ Santé Gourmet
☐ Save-a-lot Markets

Figure 3-17

When the `RepeatDirection` property is set to `Horizontal`, you get the check box items laid out in a horizontal fashion:

```
CheckBox1     CheckBox2     CheckBox3
CheckBox4     CheckBox5     CheckBox6
CheckBox7     CheckBox8     CheckBox9
CheckBox10    CheckBox11    CheckBox12
```

The RadioButton Server Control

The RadioButton server control is quite similar to the CheckBox server control. It places a radio button on your Web page. Unlike a check box, however, a single radio button on a form does not make much sense. Radio buttons are generally form elements that require at least two options. A typical set of RadioButton controls on a page takes the following construction:

```
<asp:RadioButton ID="RadioButton1" runat="server" Text="Yes" GroupName="Set1" />
<asp:RadioButton ID="RadioButton2" runat="server" Text="No" GroupName="Set1"/>
```

Figure 3-18 shows the result.

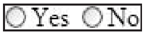

○ Yes ○ No

Figure 3-18

When you look at the code for the RadioButton control, note the standard `Text` property that places the text next to the radio button on the Web form. The more important property here is `GroupName`, which can be set in one of the RadioButton controls to match what it is set to in the other. This enables the radio buttons on the Web form to work together for the end user. How do they work together? Well, when one of the radio buttons on the form is checked, the circle associated with the item selected appears filled in. Any other filled-in circle from the same group in the collection is removed, ensuring that only one of the radio buttons in the collection is selected.

Listing 3-15 shows an example of using the RadioButton control.

Listing 3-15: Using the RadioButton server control

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub RadioButton_CheckedChanged(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        If RadioButton1.Checked = True Then
            Response.Write("You selected Visual Basic")
        Else
            Response.Write("You selected C#")
        End If
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>RadioButton control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:RadioButton ID="RadioButton1" runat="server" Text="Visual Basic"
         GroupName="LanguageChoice" OnCheckedChanged="RadioButton_CheckedChanged"
         AutoPostBack="True" />
        <asp:RadioButton ID="RadioButton2" runat="server" Text="C#"
         GroupName="LanguageChoice" OnCheckedChanged="RadioButton_CheckedChanged"
         AutoPostBack="True" />
    </div>
    </form>
</body>
</html>
```

C#

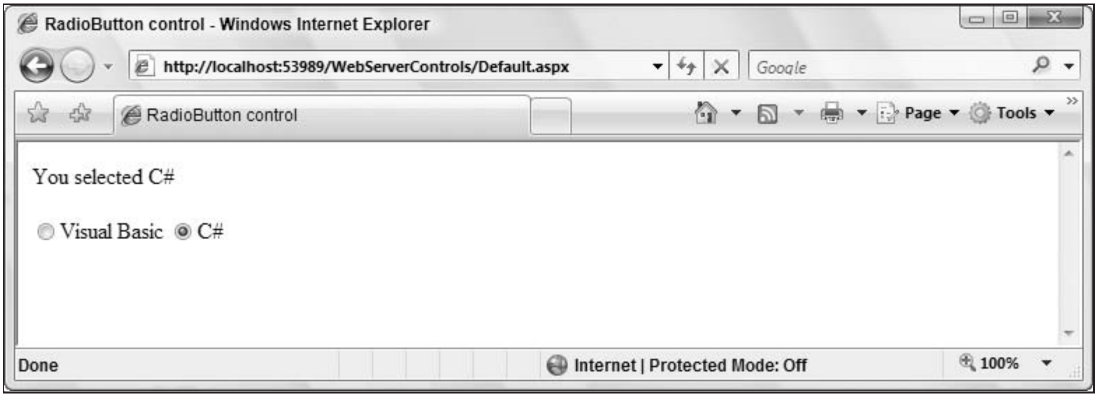
```
<%@ Page Language="C#" %>

<script runat="server">
    protected void RadioButton_CheckedChanged(object sender, EventArgs e)
    {
        if (RadioButton1.Checked == true) {
           Response.Write("You selected Visual Basic");
```

Continued

```
        }
        else {
           Response.Write("You selected C#");
        }
    }
</script>
```

Like the CheckBox, the RadioButton control has a `CheckedChanged` event that puts an `OnCheckedChanged` attribute in the control. The attribute's value points to the server-side event that is fired when a selection is made using one of the two radio buttons on the form. Remember that the `AutoPostBack` property needs to be set to `True` for this to work correctly.

Figure 3-19 shows the results.


RadioButton control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
RadioButton control
Page
Tools
Done
Internet | Protected Mode: Off
100%

You selected C#

○ Visual Basic ● C#

Figure 3-19

One advantage that the RadioButton control has over a RadioButtonList control (which is discussed next) is that it enables you to place other items (text, controls, or images) between the RadioButton controls themselves. RadioButtonList, however, is always a straight list of radio buttons on your Web page.

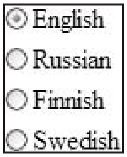
The RadioButtonList Server Control

The RadioButtonList server control lets you display a collection of radio buttons on a Web page. The RadioButtonList control is quite similar to the CheckBoxList and other list controls in that it allows you to iterate through to see what the user selected, to make counts, or to perform other actions.

A typical RadioButtonList control is written to the page in the following manner:

```
<asp:RadioButtonList ID="RadioButtonList1" runat="server">
   <asp:ListItem Selected="True">English</asp:ListItem>
   <asp:ListItem>Russian</asp:ListItem>
   <asp:ListItem>Finnish</asp:ListItem>
   <asp:ListItem>Swedish</asp:ListItem>
</asp:RadioButtonList>
```

Like the other list controls, this one uses instances of the `ListItem` object for each of the items contained in the collection. From the example, you can see that if the `Selected` property is set to `True`, one of the `ListItem` objects is selected by default when the page is generated for the first time. This produces the results shown in Figure 3-20.



● English
○ Russian
○ Finnish
○ Swedish

Figure 3-20

The `Selected` property is not required, but it is a good idea if you want the end user to make some sort of selection from this collection. Using it makes it impossible to leave the collection blank.

You can use the RadioButtonList control to check for the value selected by the end user in any of your page methods. Listing 3-16 shows a `Button1_Click` event that pushes out the value selected in the RadioButtonList collection.

Listing 3-16: Checking the value of the item selected from a RadioButtonList control

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Label1.Text = "You selected: " & _
           RadioButtonList1.SelectedItem.ToString()
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
       Label1.Text = "You selected: " +
          RadioButtonList1.SelectedItem.ToString();
    }
</script>
```

This bit of code gets at the item selected from the RadioButtonList collection of `ListItem` objects. It is how you work with other list controls that are provided in ASP.NET. The RadioButtonList also affords you access to the `RepeatColumns` and `RepeatDirection` properties (these were explained in the CheckBoxList section). You can bind this control to items that come from any of the data source controls so that you can dynamically create radio button lists on your Web pages.

Image Server Control

The Image server control enables you to work with the images that appear on your Web page from the server-side code. It is a simple server control, but it can give you the power to determine how your images are displayed on the browser screen. A typical Image control is constructed in the following manner:

```
<asp:Image ID="Image1" runat="server" ImageUrl="~/MyImage1.gif" />
```

The important property here is `ImageUrl`. It points to the file location of the image. In this case, the location is specified as the `MyImage.gif` file.

Listing 3-17 shows an example of how to dynamically change the `ImageUrl` property.

Listing 3-17: Changing the ImageUrl property dynamically

VB

```
<%@ Page Language="VB" %>

<script runat="server">
   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Image1.ImageUrl = "~/MyImage2.gif"
   End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Image control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Image ID="Image1" runat="server" ImageUrl="~/MyImage1.gif" /><br />
        <br />
        <asp:Button ID="Button1" runat="server" Text="Change Image"
         OnClick="Button1_Click" />
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
       Image1.ImageUrl = "~/MyImage2.gif";
    }
</script>
```

In this example, an image (`MyImage1.gif`) is shown in the browser when the page is loaded for the first time. When the end user clicks the button on the page, a new image (`MyImage2.gif`) is loaded in the postback process.

Special circumstances can prevent end users from viewing an image that is part of your Web page. They might be physically unable to see the image, or they might be using a text-only browser. In these cases, their browsers look for the `<img>` element's `longdesc` attribute that points to a file containing a long description of the image that is displayed.

For these cases, the Image server control includes a `DescriptionUrl` attribute. The value assigned to it is a text file that contains a thorough description of the image with which it is associated. Here is how to use it:

```
<asp:Image ID="Image1" runat="server" DescriptionUrl="~/Image01.txt"  />
```

This code produces the following results in the browser:

```
<img id="Image1" src="INETA.jpg" longdesc="Image01.txt" alt="" />
```

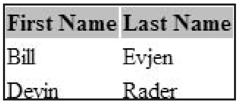
Remember that the image does not support the user clicking the image. If you want to program events based on button clicks, use the ImageButton server control discussed earlier in this chapter.

Table Server Control

Tables are one of the Web page's more common elements because the HTML `<table>` element is one possible format utilized for controlling the layout of your Web page (CSS being the other). The typical construction of the Table server control is as follows:

```
<asp:Table ID="Table1" runat="server">
   <asp:TableRow Runat="server" Font-Bold="True"
    ForeColor="Black" BackColor="Silver">
      <asp:TableHeaderCell>First Name</asp:TableHeaderCell>
      <asp:TableHeaderCell>Last Name</asp:TableHeaderCell>
   </asp:TableRow>
   <asp:TableRow>
      <asp:TableCell>Bill</asp:TableCell>
      <asp:TableCell>Evjen</asp:TableCell>
   </asp:TableRow>
  <asp:TableRow>
      <asp:TableCell>Devin</asp:TableCell>
      <asp:TableCell>Rader</asp:TableCell>
   </asp:TableRow>
</asp:Table>
```

This produces the simple three-rowed table shown in Figure 3-21.



First Name	Last Name
Bill	Evjen
Devin	Rader

Figure 3-21

You can do a lot with the Table server control. For example, you can dynamically add rows to the table, as illustrated in Listing 3-18.

Listing 3-18: Dynamically adding rows to the table

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   Dim tr As New TableRow()

   Dim fname As New TableCell()
   fname.Text = "Scott"
   tr.Cells.Add(fname)

   Dim lname As New TableCell()
   lname.Text = "Hanselman"
   tr.Cells.Add(lname)

   Table1.Rows.Add(tr)
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
   TableRow tr = new TableRow();

   TableCell fname = new TableCell();
   fname.Text = "Scott";
   tr.Cells.Add(fname);

   TableCell lname = new TableCell();
   lname.Text = "Hanselman";
   tr.Cells.Add(lname);

   Table1.Rows.Add(tr);
}
```

To add a single row to a Table control, you have to create new instances of the `TableRow` and `TableCell` objects. You create the `TableCell` objects first and then place them within a `TableRow` object that is added to a `Table` object.

The Table server control is enhanced with some extra features. One of the simpler new features is the capability to add captions to the tables on Web pages. Figure 3-22 shows a table with a caption.

To give your table a caption, simply use the new `Caption` attribute in the Table control, as illustrated in Listing 3-19.

Listing 3-19: Using the new Caption attribute

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Table Server Control</title>
</head>
```

```
<body>
    <form id="form1" runat="server">
        <asp:Table ID="Table1" runat="server"
         Caption="<b>Table 1:</b> This is an example of a caption above a table."
         BackColor="Gainsboro">
            <asp:TableRow ID="Tablerow1" Runat=server>
                <asp:TableCell ID="Tablecell1" Runat="server">Lorem ipsum dolor sit
                 amet, consectetuer adipiscing elit. Duis vel justo. Aliquam
                 adipiscing. In mattis volutpat urna. Donec adipiscing, nisl eget
                 dictum egestas, felis nulla ornare ligula, ut bibendum pede augue
                 eu augue. Sed vel risus nec urna pharetra imperdiet. Aenean
                 semper. Sed ullamcorper auctor sapien. Suspendisse luctus. Ut ac
                 nibh. Nam lorem. Aliquam dictum aliquam purus.</asp:TableCell>
            </asp:TableRow>
        </asp:Table>
    </form>
</body>
</html>
```

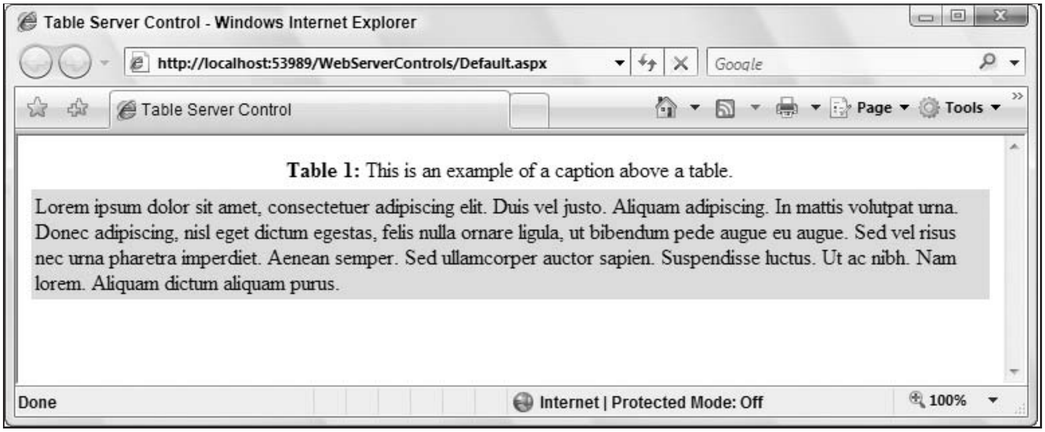


Table Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Table Server Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

Table 1: This is an example of a caption above a table.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis vel justo. Aliquam adipiscing. In mattis volutpat urna. Donec adipiscing, nisl eget dictum egestas, felis nulla ornare ligula, ut bibendum pede augue eu augue. Sed vel risus nec urna pharetra imperdiet. Aenean semper. Sed ullamcorper auctor sapien. Suspendisse luctus. Ut ac nibh. Nam lorem. Aliquam dictum aliquam purus.

Figure 3-22

By default, the caption is placed at the top center of the table, but you can control where it is placed by using another new attribute — `CaptionAlign`. Its possible settings include `Bottom`, `Left`, `NotSet`, `Right`, and `Top`.

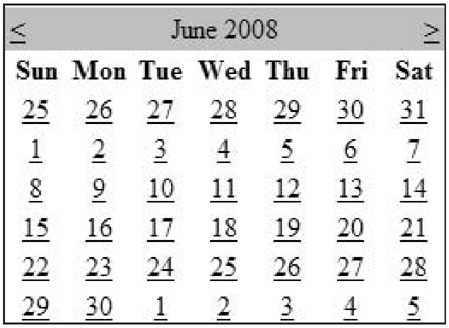
In the past, an `<asp:Table>` element contained any number of `<asp:TableRow>` elements. Now you have some additional elements that can be nested within the `<asp:Table>` element. These new elements include `<asp:TableHeaderRow>` and `<asp:TableFooterRow>`. They add either a header or footer to your table, enabling you to use the Table server control to page through lots of data but still retain some text in place to indicate the type of data being handled. This is quite a powerful feature when you work with mobile applications that dictate that sometimes end users can move through only a few records at a time.

The Calendar Server Control

The Calendar server control is a rich control that enables you to place a full-featured calendar directly on your Web pages. It allows for a high degree of customization to ensure that it looks and behaves in a unique manner. The Calendar control, in its simplest form, is coded in the following manner:

```
<asp:Calendar ID="Calendar1" runat="server">
</asp:Calendar>
```

This code produces a calendar on your Web page without any styles added, as shown in Figure 3-23.



<			June 2008			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

Figure 3-23

Making a Date Selection from the Calendar Control

The calendar allows you to navigate through the months of the year and to select specific days in the exposed month. A simple application that enables the user to select a day of the month is shown in Listing 3-20.

Listing 3-20: Selecting a single day in the Calendar control

VB

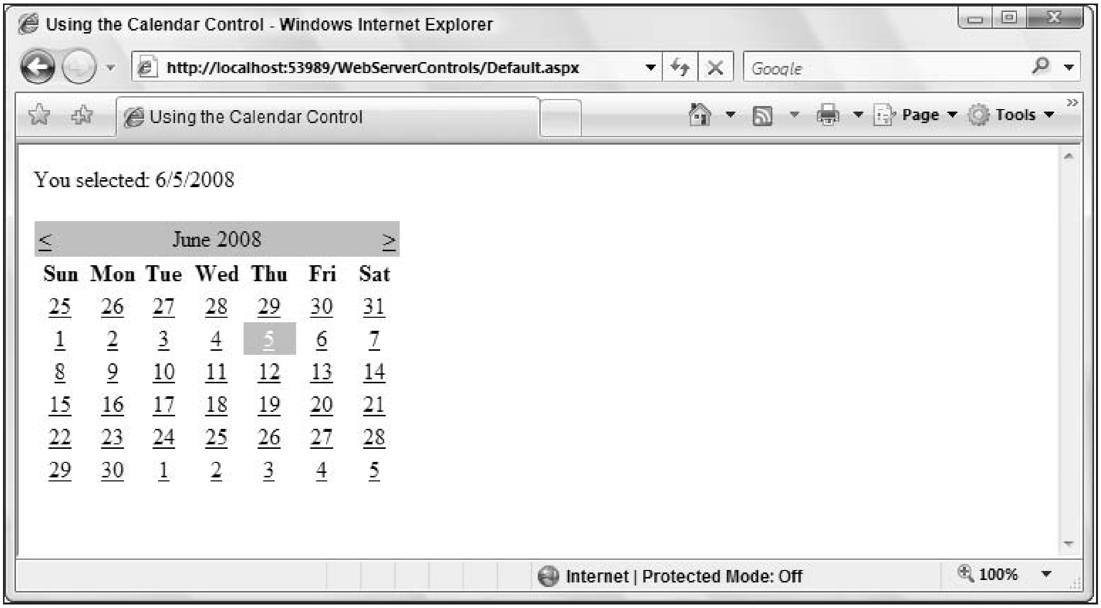
```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Calendar1_SelectionChanged(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        Response.Write("You selected: " & _
          Calendar1.SelectedDate.ToShortDateString())
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Using the Calendar Control</title>
</head>
```

```
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Calendar ID="Calendar1" runat="server"
         OnSelectionChanged="Calendar1_SelectionChanged">
        </asp:Calendar>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Calendar1_SelectionChanged(object sender, EventArgs e)
    {
        Response.Write("You selected: " +
          Calendar1.SelectedDate.ToShortDateString());
    }
</script>
```

Running this application pulls up the calendar in the browser. The end user can then select a single date in it. After a date is selected, the `Calendar1_SelectionChanged` event is triggered and makes use of the `OnSelectionChange` attribute. This event writes the value of the selected date to the screen. The result is shown in Figure 3-24.


Using the Calendar Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Using the Calendar Control
Page
Tools
Internet | Protected Mode: Off
100%

You selected: 6/5/2008

<			June 2008			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

Figure 3-24

Choosing a Date Format to Output from the Calendar

When you use the `Calendar1_SelectionChanged` event, the selected date is written out using the `ToShortDateString()` method. The Calendar control also allows you to write out the date in a number of other formats, as detailed in the following list:

- ☐ `ToFileTime`: Converts the selection to the local operating system file time: `128571156000000000`.
- ☐ `ToFileTimeUtc`: Converts the selection to the operating system file time, but instead of using the local time zone, the UTC time is used: `128570976000000000`.
- ☐ `ToLocalTime`: Converts the current coordinated universal time (UTC) to local time: `6/4/2008 7:00:00 PM`.
- ☐ `ToLongDateString`: Converts the selection to a human-readable string in a long format: `Thursday, June 05, 2008`.
- ☐ `ToLongTimeString`: Converts the selection to a time value (no date is included) of a long format: `12:00:00 AM`.
- ☐ `ToOADate`: Converts the selection to an OLE Automation date equivalent: `39604`.
- ☐ `ToShortDateString`: Converts the selection to a human-readable string in a short format: `6/4/2008`.
- ☐ `ToShortTimeString`: Converts the selection to a time value (no date is included) in a short format: `12:00 AM`.
- ☐ `ToString`: Converts the selection to the following: `6/4/2008 12:00:00 AM`.
- ☐ `ToUniversalTime`: Converts the selection to universal time (UTC): `6/4/2008 6:00:00 AM`.

Making Day, Week, or Month Selections

By default, the Calendar control enables you to make single day selections. You can use the `SelectionMode` property to change this behavior to allow your users to make week or month selections from the calendar instead. The possible values of this property include `Day`, `DayWeek`, `DayWeekMonth`, and `None`.

The `Day` setting enables you to click a specific day in the calendar to highlight it (this is the default). When you use the setting of `DayWeek`, you can still make individual day selections, but you can also click the arrow next to the week (see Figure 3-25) to make selections that consist of an entire week. Using the setting of `DayWeekMonth` lets users make individual day selections or week selections. A new arrow appears in the upper-left corner of the calendar that enables users to select an entire month (also shown in Figure 3-25). A setting of `None` means that it is impossible for the end user to make any selections, which is useful for calendars on your site that are informational only.

Working with Date Ranges

Even if an end user makes a selection that encompasses an entire week or an entire month, you get back from the selection only the first date of this range. If, for example, you allow users to select an entire month and one selects July 2008, what you get back (using `ToShortDateString()`) is `7/1/2008` — the first date in the date range of the selection. That might work for you, but if you require all the dates in the selected range, Listing 3-21 shows you how to get them.

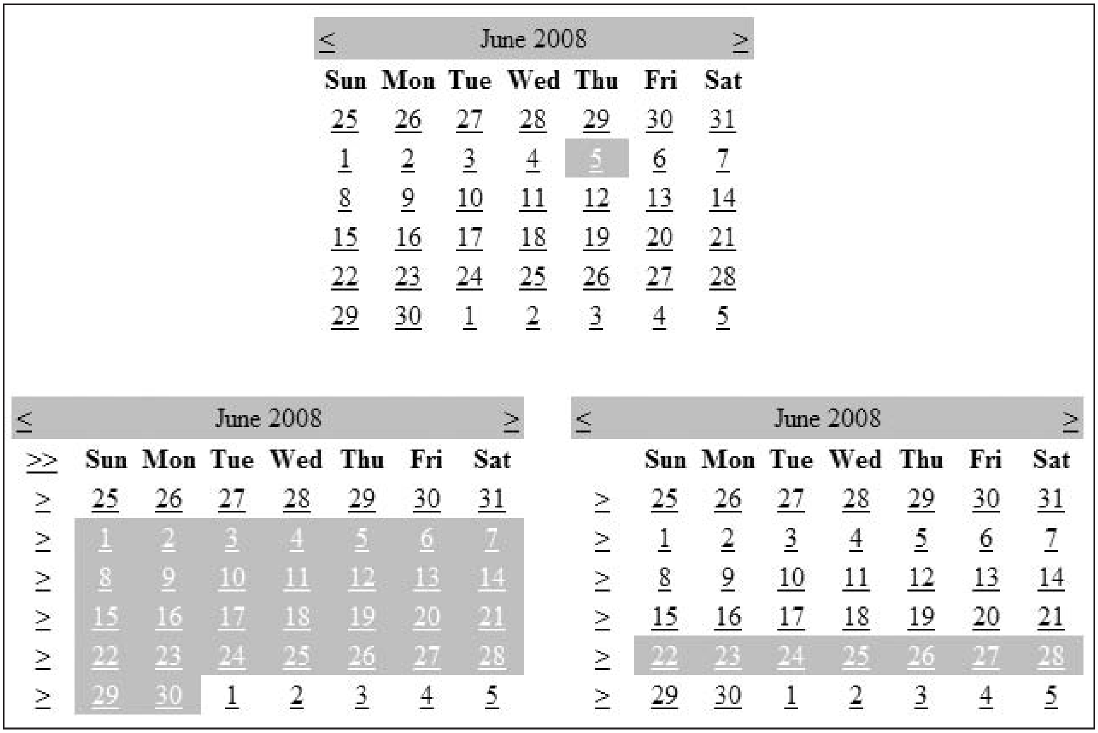


Figure 3-25

Listing 3-21: Retrieving a range of dates from a selection

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Calendar1_SelectionChanged(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        Label1.Text = "<b><u>You selected the following date/dates:</u></b><br>"

        For i As Integer = 0 To (Calendar1.SelectedDates.Count - 1)
            Label1.Text += Calendar1.SelectedDates.Item(i).ToShortDateString() & _
              "<br>"
        Next
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Using the Calendar Control</title>
</head>
```

Continued

```
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Calendar ID="Calendar1" runat="server"
         OnSelectionChanged="Calendar1_SelectionChanged"
         SelectionMode="DayWeekMonth">
        </asp:Calendar><p>
        <asp:Label ID="Label1" runat="server"></asp:Label></p>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Calendar1_SelectionChanged(object sender, EventArgs e)
    {
       Label1.Text = "<b><u>You selected the following date/dates:</u></b><br>";

        for (int i=0; i<Calendar1.SelectedDates.Count; i++) {
            Label1.Text += Calendar1.SelectedDates[i].ToShortDateString() +
              "<br>";
        }
    }
</script>
```

In this example, the Calendar control lets users make selections that can be an individual day, a week, or even a month. Using a `For Next` loop, you iterate through a selection by using the `SelectedDates.Count` property. The code produces the results shown in Figure 3-26.

You can get just the first day of the selection by using the following:

VB

```
Calendar1.SelectedDates.Item(0).ToShortDateString()
```

C#

```
Calendar1.SelectedDates[0].ToShortDateString();
```

And you can get the last date in the selected range by using:

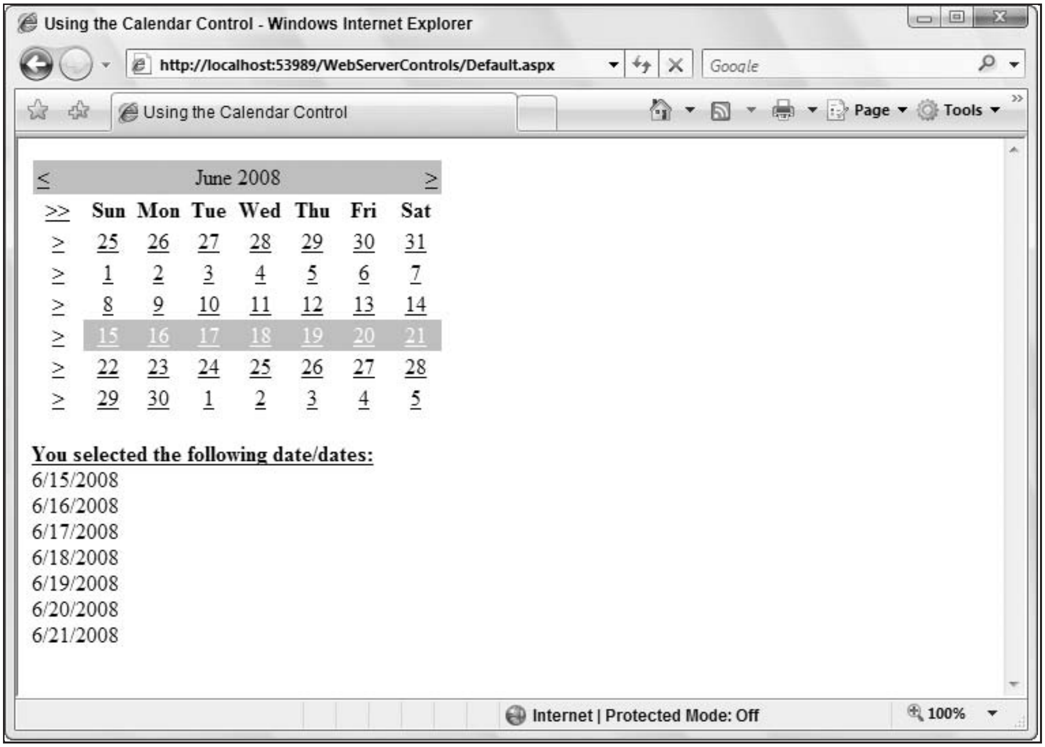
VB

```
Calendar1.SelectedDates.Item(Calendar1.SelectedDates.Count-1).ToShortDateString()
```

C#

```
Calendar1.SelectedDates[Calendar1.SelectedDates.Count-1].ToShortDateString();
```

As you can see, this is possible using the `Count` property of the `SelectedDates` object.


Using the Calendar Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Using the Calendar Control
Page
Tools
Internet | Protected Mode: Off
100%

< June 2008 >

>>	Sun	Mon	Tue	Wed	Thu	Fri	Sat
>	25	26	27	28	29	30	31
>	1	2	3	4	5	6	7
>	8	9	10	11	12	13	14
>	15	16	17	18	19	20	21
>	22	23	24	25	26	27	28
>	29	30	1	2	3	4	5

You selected the following date/dates:
6/15/2008
6/16/2008
6/17/2008
6/18/2008
6/19/2008
6/20/2008
6/21/2008

Figure 3-26

Modifying the Style and Behavior of Your Calendar

There is a lot to the Calendar control — definitely more than can be covered in this chapter. One nice thing about the Calendar control is the ease of extensibility that it offers. Begin exploring new ways to customize this control further by looking at one of the easiest ways to change it — applying a style to the control.

Using Visual Studio, you can give the controls a new look-and-feel from the Design view of the page you are working with. Highlight the Calendar control and open the control's smart tag to see the Auto Format link. That gives you a list of available styles that can be applied to your Calendar control.

> *The Calendar control is not alone in this capability. Many other rich controls offer a list of styles. You can always find this capability in the control's smart tag.*

Some of the styles are shown in Figure 3-27.

In addition to changing the style of the Calendar control, you can work with the control during its rendering process. The Calendar control includes an event called `DayRender` that allows you to control how a single date or all the dates in the calendar are rendered. Listing 3-22 shows an example of how to change one of the dates being rendered in the calendar.

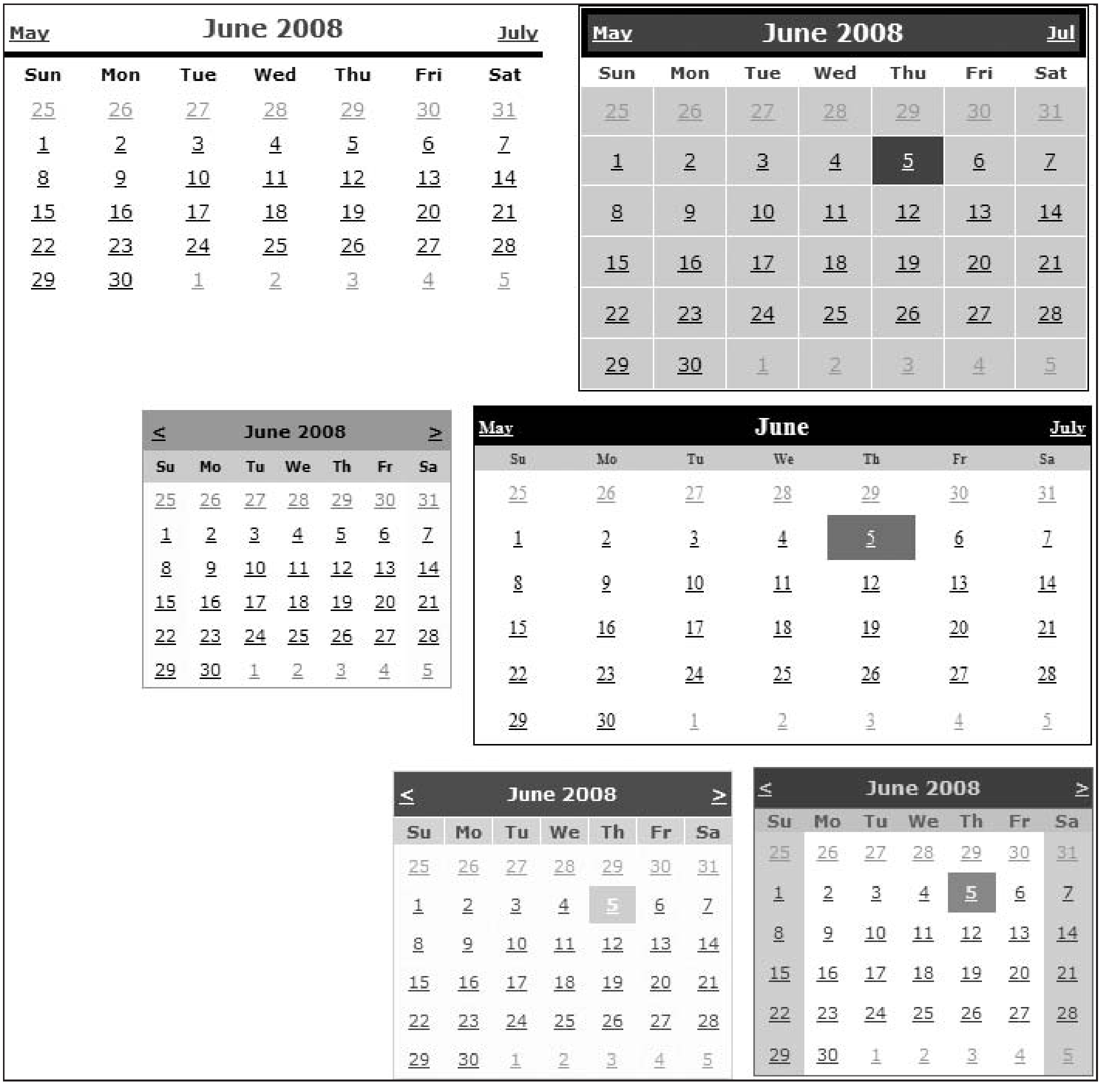

May
June 2008
July
May
June 2008
Jul
<
June 2008
>
May
June
July
<
June 2008
>
<
June 2008
>

Figure 3-27

Listing 3-22: Controlling how a day is rendered in the Calendar

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Calendar1_DayRender(ByVal sender As Object, _
      ByVal e As System.Web.UI.WebControls.DayRenderEventArgs)
        e.Cell.VerticalAlign = VerticalAlign.Top

        If (e.Day.DayNumberText = "25") Then
```

```
            e.Cell.Controls.Add(New LiteralControl("<p>User Group Meeting!</p>"))
            e.Cell.BorderColor = Drawing.Color.Black
            e.Cell.BorderWidth = 1
            e.Cell.BorderStyle = BorderStyle.Solid
            e.Cell.BackColor = Drawing.Color.LightGray
        End If
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Using the Calendar Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Calendar ID="Calendar1" runat="server"
         OnDayRender="Calendar1_DayRender" Height="190px" BorderColor="White"
         Width="350px" ForeColor="Black" BackColor="White" BorderWidth="1px"
         NextPrevFormat="FullMonth" Font-Names="Verdana" Font-Size="9pt">
            <SelectedDayStyle ForeColor="White"
             BackColor="#333399"></SelectedDayStyle>
            <OtherMonthDayStyle ForeColor="#999999"></OtherMonthDayStyle>
            <TodayDayStyle BackColor="#CCCCCC"></TodayDayStyle>
            <NextPrevStyle ForeColor="#333333" VerticalAlign="Bottom"
             Font-Size="8pt" Font-Bold="True"></NextPrevStyle>
            <DayHeaderStyle Font-Size="8pt" Font-Bold="True"></DayHeaderStyle>
            <TitleStyle ForeColor="#333399" BorderColor="Black" Font-Size="12pt"
                Font-Bold="True" BackColor="White" BorderWidth="4px">
            </TitleStyle>
        </asp:Calendar>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Calendar1_DayRender(object sender, DayRenderEventArgs e)
    {
        e.Cell.VerticalAlign = VerticalAlign.Top;

        if (e.Day.DayNumberText == "25")
        {
            e.Cell.Controls.Add(new LiteralControl("<p>User Group Meeting!</p>"));
            e.Cell.BorderColor = System.Drawing.Color.Black;
            e.Cell.BorderWidth = 1;
            e.Cell.BorderStyle = BorderStyle.Solid;
            e.Cell.BackColor = System.Drawing.Color.LightGray;
        }

    }
</script>
```

In this example, you use a Calendar control with a little style to it. When the page is built and run in the browser, you can see that the 25th of every month in the calendar has been changed by the code in the `Calendar1_DayRender` event. The calendar is shown in Figure 3-28.

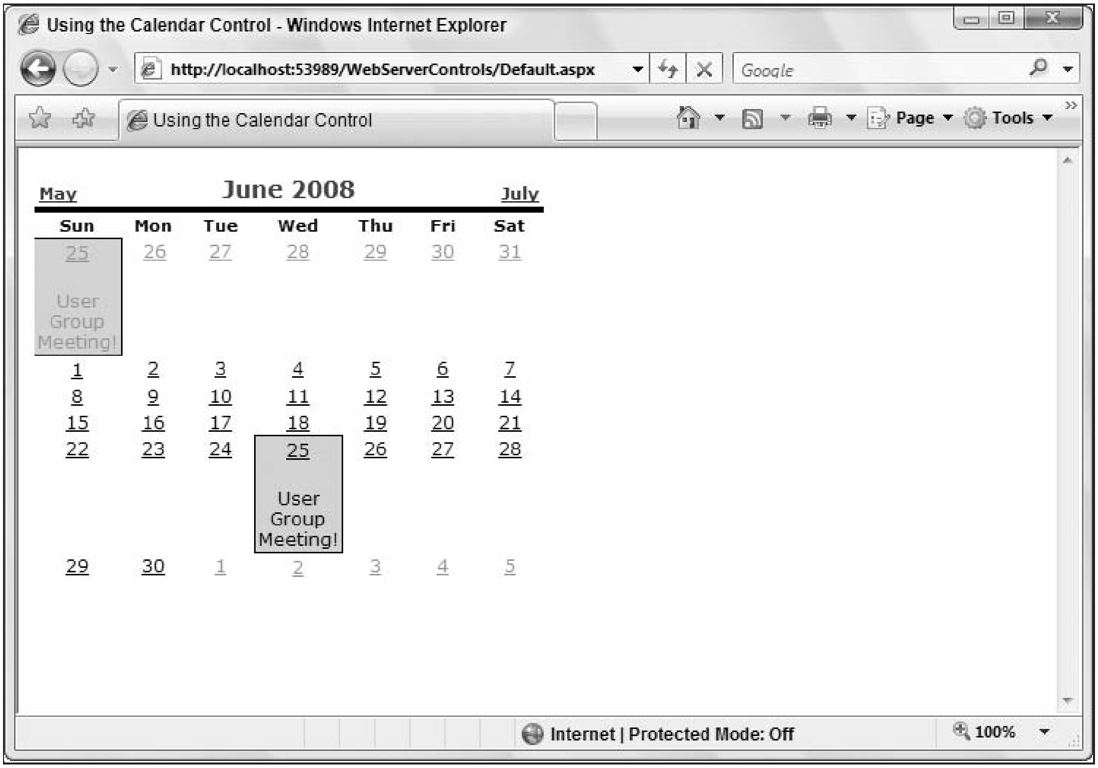

Using the Calendar Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Using the Calendar Control
Page
Tools
May
June 2008
July
Sun
Mon
Tue
Wed
Thu
Fri
Sat
25
26
27
28
29
30
31
User Group Meeting!
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
User Group Meeting!
29
30
1
2
3
4
5
Internet | Protected Mode: Off
100%

Figure 3-28

The Calendar control in this example adds an `OnDayRender` attribute that points to the `Calendar1_DayRender` event. The method is run for each of the days rendered in the calendar. The class constructor shows that you are not working with the typical `System.EventArgs` class, but instead with the `DayRenderEventArgs` class. It gives you access to each of the days rendered in the calendar.

The two main properties from the `DayRenderEventArgs` class are `Cell` and `Day`. The `Cell` property gives you access to the space in which the day is being rendered, and the `Day` property gives you access to the specific date being rendered in the cell.

From the actions being taken in the `Calendar1_DayRender` event, you can see that both properties are used. First, the `Cell` property sets the vertical alignment of the cell to `Top`. If it didn't, the table might look a little strange when one of the cells has content. Next, a check is made to see if the day being rendered (checked with the `Day` property) is the 25th of the month. If it is, the `If Then` statement runs using the `Cell` property to change the styling of just that cell. The styling change adds a control, as well as makes changes to the border and color of the cell.

As you can see, working with individual dates in the calendar is fairly straightforward. You can easily give them the content and appearance you want.

A nice feature of the `Day` property is that you can turn off the option to select a particular date or range of dates by setting the `Day` property's `IsSelectable` property to `False`:

VB

```
If (e.Day.Date < DateTime.Now) Then
   e.Day.IsSelectable = False
End If
```

C#

```
if (e.Day.Date < DateTime.Now) {
   e.Day.IsSelectable = false;
}
```

AdRotator Server Control

Although Web users find ads rather annoying, advertising continues to be prevalent everywhere on the Web. With the AdRotator control, you can configure your application to show a series of advertisements to the end users. With this control, you can use advertisement data from sources other than the standard XML file that was used with the previous versions of this control.

If you are using an XML source for the ad information, first create an XML advertisement file. The advertisement file allows you to incorporate some new elements that give you even more control over the appearance and behavior of your ads. Listing 3-23 shows an example of an XML advertisement file.

Listing 3-23: The XML advertisement file

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements
 xmlns="http://schemas.microsoft.com/AspNet/AdRotator-Schedule-File">
 <Ad>
    <ImageUrl>book1.gif</ImageUrl>
    <NavigateUrl>http://www.wrox.com</NavigateUrl>
    <AlternateText>Visit Wrox Today!</AlternateText>
    <Impressions>50</Impressions>
    <Keyword>VB.NET</Keyword>
 </Ad>
 <Ad>
    <ImageUrl>book2.gif</ImageUrl>
    <NavigateUrl>http://www.wrox.com</NavigateUrl>
    <AlternateText>Visit Wrox Today!</AlternateText>
    <Impressions>50</Impressions>
    <Keyword>XML</Keyword>
 </Ad>
</Advertisements>
```

This XML file, used for storing information about the advertisements that appear in your application, has just a few elements detailed in the following table. Remember that all elements are optional.

Element	Description
`ImageUrl`	Takes a string value that indicates the location of the image to use.
`NavigateUrl`	Takes a string value that indicates the URL to post to when the image is clicked.
`AlternateText`	Takes a string value that is used for display either if images are turned off in the client's browser or if the image is not found.
`Impressions`	Takes a numerical value that indicates the likelihood of the image being selected for display.
`Keyword`	Takes a string value that sets the category of the image in order to allow for the filtering of ads.

Now that the XML advertisement file is in place, you can simply use the AdRotator control to read from this file. Listing 3-24 shows an example of this in action.

Listing 3-24: Using the AdRotator control as a banner ad

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>AdRotator Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:AdRotator ID="AdRotator1" runat="server"
         AdvertisementFile="MyAds.xml" />
        <p>Lorem ipsum dolor sit
        amet, consectetuer adipiscing elit. Duis vel justo. Aliquam
        adipiscing. In mattis volutpat urna. Donec adipiscing, nisl eget
        dictum egestas, felis nulla ornare ligula, ut bibendum pede augue
        eu augue. Sed vel risus nec urna pharetra imperdiet. Aenean
        semper. Sed ullamcorper auctor sapien. Suspendisse luctus. Ut ac
        nibh. Nam lorem. Aliquam dictum aliquam purus.</p>
    </form>
</body>
</html>
```

The example shows the ad specified in the XML advertisement file as a banner ad at the top of the page.

You are not required to place all your ad information in the XML advertisement file. Instead, you can use another data source to which you bind the AdRotator. For instance, you bind the AdRotator to a `SqlDataSource` object that is retrieving the ad information from SQL Server in the following fashion:

```
<asp:AdRotator ID="AdRotator1" runat="server"
 DataSourceId="SqlDataSource1" AlternateTextField="AlternateTF"
 ImageUrlField="Image" NavigateUrlField="NavigateUrl" />
```

The `AlternateTextField`, `ImageUrlField`, and `NavigateUrlField` properties point to the column names that are used in SQL Server for those items.

The Xml Server Control

The Xml server control provides a means of getting XML and transforming it using an XSL style sheet. The Xml control can work with your XML in a couple of different ways. The simplest method is by using the construction shown in Listing 3-25. This control is covered in more detail in Chapter 10.

Listing 3-25: Displaying an XML document

```
<asp:Xml ID="Xml1" runat="server" DocumentSource="~/MyXMLFile.xml"
 TransformSource="MyXSLFile.xslt"></asp:Xml>
```

This method takes only a couple of attributes to make it work: `DocumentSource`, which points to the path of the XML file, and `TransformSource`, which provides the XSLT file to use in transforming the XML document.

The other way to use the Xml server control is to load the XML into an object and then pass the object to the Xml control, as illustrated in Listing 3-26.

Listing 3-26: Loading the XML file to an object before providing it to the Xml control

VB

```
Dim MyXmlDoc as XmlDocument = New XmlDocument()
MyXmlDoc.Load(Server.MapPath("Customers.xml"))

Dim MyXslDoc As XslCompiledTransform = New XslCompiledTransform()
MyXslDoc.Load(Server.MapPath("CustomersSchema.xslt"))

Xml1.Document = MyXmlDoc
Xml1.Transform = MyXslDoc
```

C#

```
XmlDocument MyXmlDoc = new XmlDocument();
MyXmlDoc.Load(Server.MapPath("Customers.xml"));

XslCompiledTransform MyXsltDoc = new XslCompiledTransform();
MyXsltDoc.Load(Server.MapPath("CustomersSchema.xslt"));

Xml1.Document = MyXmlDoc;
Xml1.Transform = MyXslDoc;
```

To make this work, you have to ensure that the `System.Xml` and `System.Xml.Xsl` namespaces are imported into your page. The example loads both the XML and XSL files and then assigns these files as the values of the `Document` and `Transform` properties.

Panel Server Control

The Panel server control encapsulates a set of controls you can use to manipulate or lay out your ASP.NET pages. It is basically a wrapper for other controls, enabling you to take a group of server controls along with other elements (such as HTML and images) and turn them into a single unit.

The advantage of using the Panel control to encapsulate a set of other elements is that you can manipulate these elements as a single unit using one attribute set in the Panel control itself. For example, setting the `Font-Bold` attribute to `True` causes each item within the Panel control to adopt this attribute.

The new addition to the Panel control is the capability to scroll with scrollbars that appear automatically depending on the amount of information that Panel control holds. You can even specify how the scrollbars should appear.

For an example of using scrollbars, look at a long version of the Lorem Ipsum text (found at `www.lipsum.com`) and place that text within the Panel control, as shown in Listing 3-27.

Listing 3-27: Using the new scrollbar feature with the Panel server control

```
<%@ Page Language="VB" %>

<html>
<head runat="server">
    <title>Panel Server Control Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Panel ID="Panel1" runat="server" Height="300" Width="300"
         ScrollBars="auto">
           <p>Lorem ipsum dolor sit amet...</p>
        </asp:Panel>
    </form>
</body>
</html>
```

By assigning values to the `Height` and `Width` attributes of the Panel server control and using the `ScrollBars` attribute (in this case, set to `Auto`), you can display the information it contains within the defined area using scrollbars (see Figure 3-29).

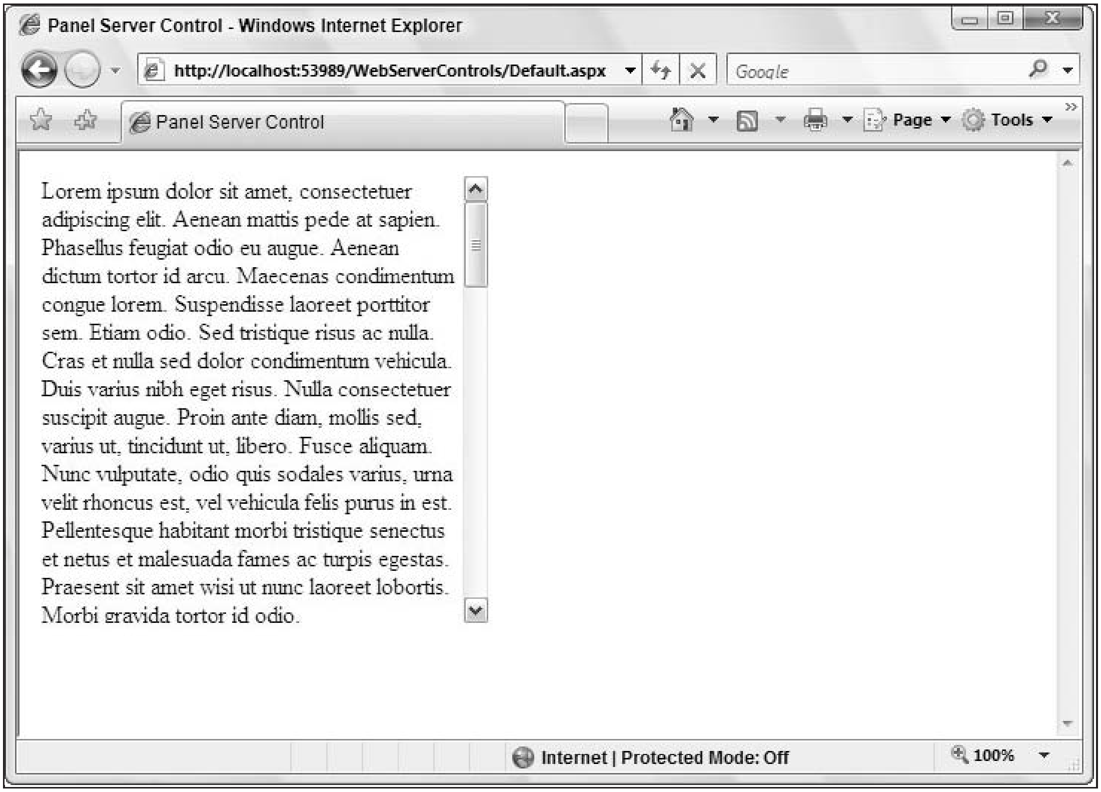
As you can see, a single vertical scrollbar has been added to the set area of 300 × 300 pixels. The Panel control wraps the text by default as required. To change this behavior, use the new `Wrap` attribute, which takes a `Boolean` value:

```
<asp:Panel ID="Panel1" runat="server"
 Height="300" Width="300" ScrollBars="Auto"
 Wrap="False" />
```

Turning off wrapping may cause the horizontal scrollbar to turn on (depending on what is contained in the panel section).

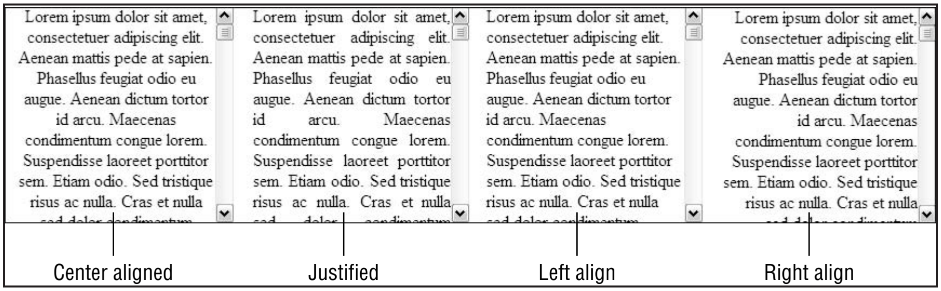
If you do not want to let the ASP.NET engine choose which scrollbars to activate, you can actually make that decision by using the `ScrollBars` attribute. In addition to `Auto`, its values include `None`, `Horizontal`, `Vertical`, and `Both`.

Another interesting attribute that enables you to change the behavior of the Panel control is `HorizontalAlign`. It enables you to set how the content in the Panel control is horizontally aligned. The possible values of this attribute include `NotSet`, `Center`, `Justify`, `Left`, and `Right`. Figure 3-30 shows a collection of Panel controls with different horizontal alignments.


Panel Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Panel Server Control
Page
Tools
Internet | Protected Mode: Off
100%

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean mattis pede at sapien. Phasellus feugiat odio eu augue. Aenean dictum tortor id arcu. Maecenas condimentum congue lorem. Suspendisse laoreet porttitor sem. Etiam odio. Sed tristique risus ac nulla. Cras et nulla sed dolor condimentum vehicula. Duis varius nibh eget risus. Nulla consectetuer suscipit augue. Proin ante diam, mollis sed, varius ut, tincidunt ut, libero. Fusce aliquam. Nunc vulputate, odio quis sodales varius, urna velit rhoncus est, vel vehicula felis purus in est. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Praesent sit amet wisi ut nunc laoreet lobortis. Morbi gravida tortor id odio.

Figure 3-29



Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean mattis pede at sapien. Phasellus feugiat odio eu augue. Aenean dictum tortor id arcu. Maecenas condimentum congue lorem. Suspendisse laoreet porttitor sem. Etiam odio. Sed tristique risus ac nulla. Cras et nulla

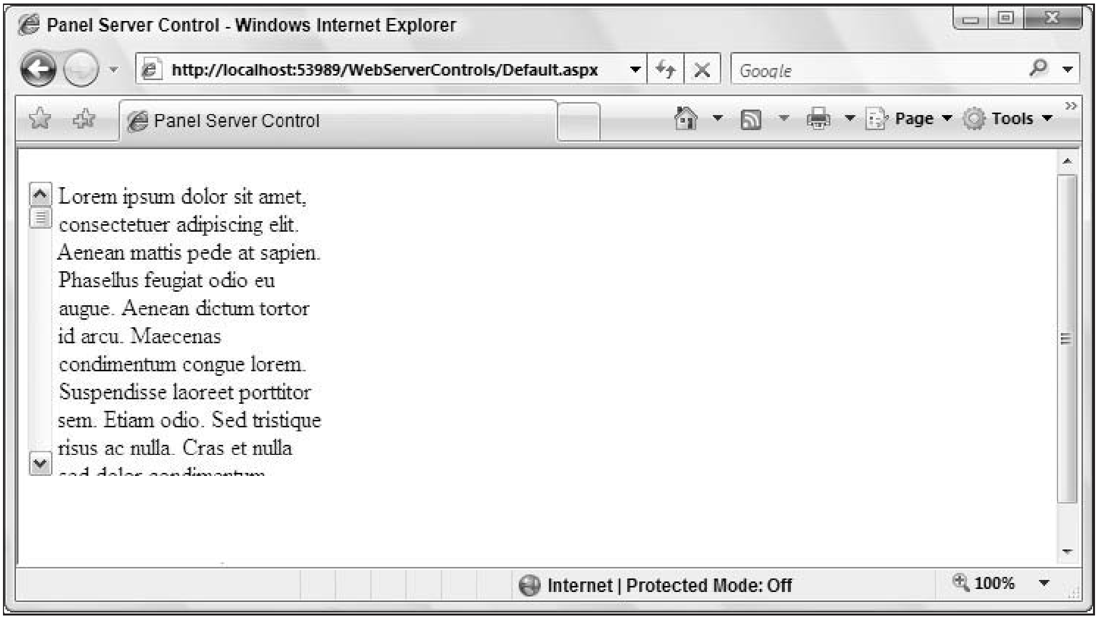
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean mattis pede at sapien. Phasellus feugiat odio eu augue. Aenean dictum tortor id arcu. Maecenas condimentum congue lorem. Suspendisse laoreet porttitor sem. Etiam odio. Sed tristique risus ac nulla. Cras et nulla

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean mattis pede at sapien. Phasellus feugiat odio eu augue. Aenean dictum tortor id arcu. Maecenas condimentum congue lorem. Suspendisse laoreet porttitor sem. Etiam odio. Sed tristique risus ac nulla. Cras et nulla

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean mattis pede at sapien. Phasellus feugiat odio eu augue. Aenean dictum tortor id arcu. Maecenas condimentum congue lorem. Suspendisse laoreet porttitor sem. Etiam odio. Sed tristique risus ac nulla. Cras et nulla

Figure 3-30

It is also possible to move the vertical scrollbar to the left side of the Panel control by using the `Direction` attribute. `Direction` can be set to `NotSet`, `LeftToRight`, and `RightToLeft`. A setting of `RightToLeft` is ideal when you are dealing with languages that are written from right to left (some Asian languages, for example). However, that setting also moves the scrollbar to the left side of the Panel control. If the scrollbar is moved to the left side and the `HorizontalAlign` attribute is set to `Left`, your content resembles Figure 3-31.


Panel Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Panel Server Control
Page
Tools
Internet | Protected Mode: Off
100%

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean mattis pede at sapien. Phasellus feugiat odio eu augue. Aenean dictum tortor id arcu. Maecenas condimentum congue lorem. Suspendisse laoreet porttitor sem. Etiam odio. Sed tristique risus ac nulla. Cras et nulla

Figure 3-31

The PlaceHolder Server Control

The PlaceHolder server control works just as its name implies — it is a placeholder for you to interject objects dynamically into your page. Think of it as a marker with which you can add other controls. The capability to add controls to a page at a specific point also works with the Panel control.

To see how it works, insert a PlaceHolder control into your page and then add controls to it from your server-side code in the manner shown in Listing 3-28.

Listing 3-28: Using PlaceHolder to add controls to a page dynamically

VB

```
Dim NewLabelControl As New Label()
NewLabelControl.Text = "Hello there"
PlaceHolder1.Controls.Add(NewLabelControl)
```

C#

```
Label NewLabelControl = new Label();
NewLabelControl.Text = "Hello there";
PlaceHolder1.Controls.Add(NewLabelControl);
```

This example creates a new instance of a Label control and populates it with a value before it is added to the PlaceHolder control. You can add more than one control to a single instance of a PlaceHolder control.

BulletedList Server Control

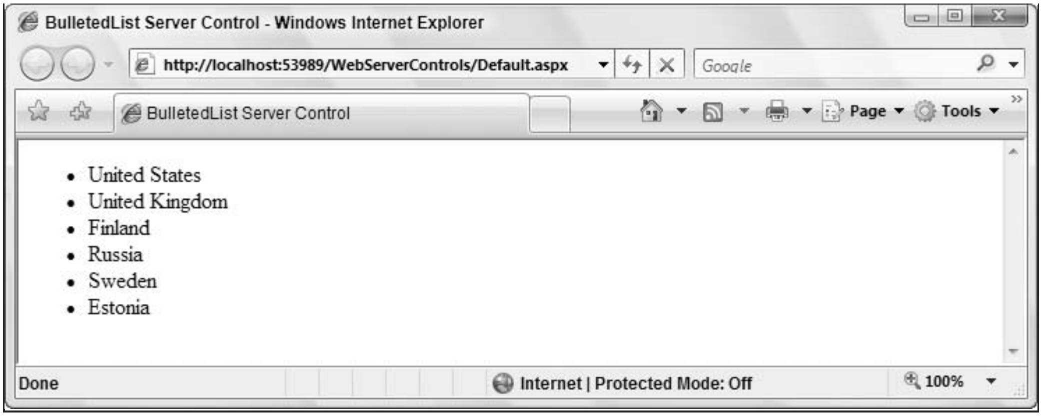
One common HTML Web page element is a collection of items in a bulleted list. The BulletedList server control is meant to display a bulleted list of items easily in an ordered (using the HTML `<ol>` element) or unordered (using the HTML `<ul>` element) fashion. In addition, the control can determine the style used for displaying the list.

The BulletedList control can be constructed of any number of `<asp:ListItem>` controls or can be data-bound to a data source of some kind and populated based upon the contents retrieved. Listing 3-29 shows a bulleted list in its simplest form.

Listing 3-29: A simple BulletedList control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>BulletedList Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:BulletedList ID="Bulletedlist1" runat="server">
            <asp:ListItem>United States</asp:ListItem>
            <asp:ListItem>United Kingdom</asp:ListItem>
            <asp:ListItem>Finland</asp:ListItem>
            <asp:ListItem>Russia</asp:ListItem>
            <asp:ListItem>Sweden</asp:ListItem>
            <asp:ListItem>Estonia</asp:ListItem>
        </asp:BulletedList>
    </form>
</body>
</html>
```

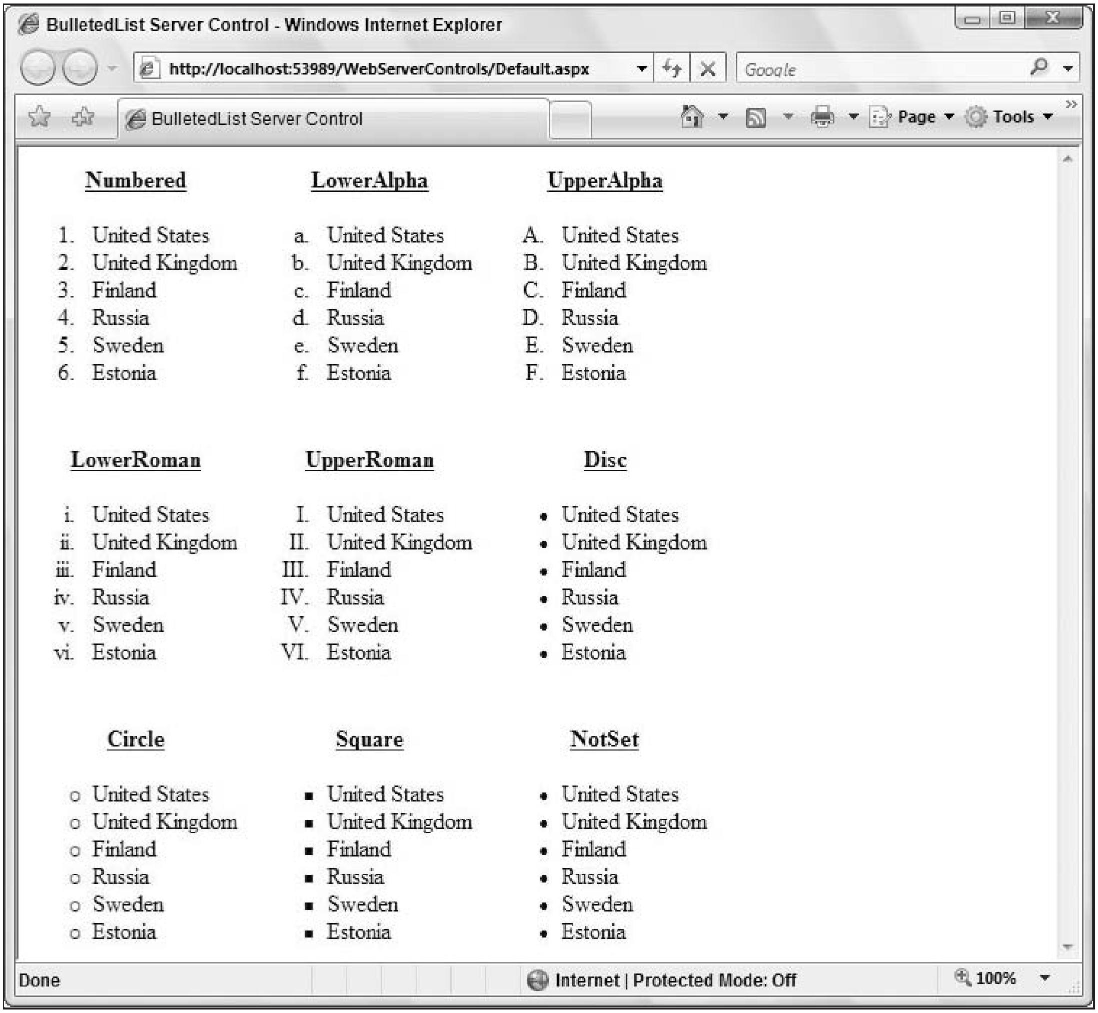
The use of the `<asp:BulletedList>` element, along with `<asp:ListItem>` elements, produces a simple bulleted list output like the one shown in Figure 3-32.


BulletedList Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
BulletedList Server Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

- United States
- United Kingdom
- Finland
- Russia
- Sweden
- Estonia

Figure 3-32

The BulletedList control also enables you to easily change the style of the list with just one or two attributes. The `BulletStyle` attribute changes the style of the bullet that precedes each line of the list. It has possible values of `Numbered`, `LowerAlpha`, `UpperAlpha`, `LowerRoman`, `UpperRoman`, `Disc`, `Circle`, `Square`, `NotSet`, and `CustomImage`. Figure 3-33 shows examples of these styles (minus the `CustomImage` setting that enables you to use any image of your choice).


BulletedList Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
BulletedList Server Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

Numbered

1. United States
2. United Kingdom
3. Finland
4. Russia
5. Sweden
6. Estonia

LowerAlpha

a. United States
b. United Kingdom
c. Finland
d. Russia
e. Sweden
f. Estonia

UpperAlpha

A. United States
B. United Kingdom
C. Finland
D. Russia
E. Sweden
F. Estonia

LowerRoman

i. United States
ii. United Kingdom
iii. Finland
iv. Russia
v. Sweden
vi. Estonia

UpperRoman

I. United States
II. United Kingdom
III. Finland
IV. Russia
V. Sweden
VI. Estonia

Disc

- United States
- United Kingdom
- Finland
- Russia
- Sweden
- Estonia

Circle

○ United States
○ United Kingdom
○ Finland
○ Russia
○ Sweden
○ Estonia

Square

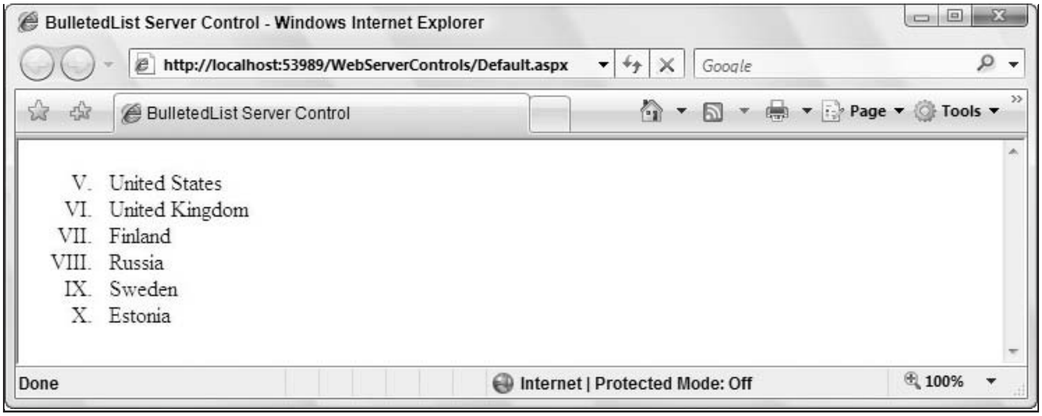
▪ United States
▪ United Kingdom
▪ Finland
▪ Russia
▪ Sweden
▪ Estonia

NotSet

- United States
- United Kingdom
- Finland
- Russia
- Sweden
- Estonia

Figure 3-33

You can change the starting value of the first item in any of the numbered styles (`Numbered`, `LowerAlpha`, `UpperAlpha`, `LowerRoman`, `UpperRoman`) by using the `FirstBulletNumber` attribute. If you set the attribute's value to `5` when you use the `UpperRoman` setting, for example, you get the format illustrated in Figure 3-34.


BulletedList Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
BulletedList Server Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

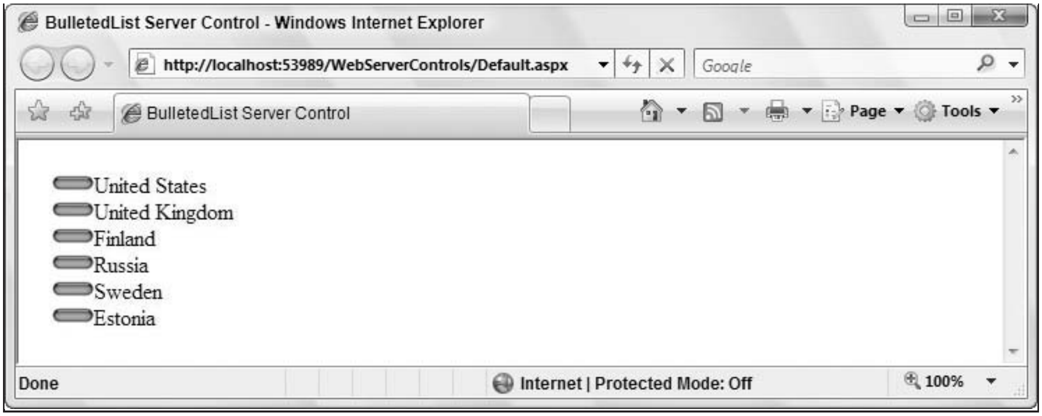
V. United States
VI. United Kingdom
VII. Finland
VIII. Russia
IX. Sweden
X. Estonia

Figure 3-34

To employ images as bullets, use the `CustomImage` setting in the BulletedList control. You must also use the `BulletImageUrl` attribute in the following manner:

```
<asp:BulletedList ID="Bulletedlist1" runat="server"
 BulletStyle="CustomImage" BulletImageUrl="~/myImage.gif">
```

Figure 3-35 shows an example of image bullets.


BulletedList Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
BulletedList Server Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

United States
United Kingdom
Finland
Russia
Sweden
Estonia

Figure 3-35

The BulletedList control has an attribute called `DisplayMode`, which has three possible values: `Text`, `HyperLink`, and `LinkButton`. `Text` is the default and has been used so far in the examples. Using `Text` means that the items in the bulleted list are laid out only as text. `HyperLink` means that each of the items is turned into a hyperlink — any user clicking the link is redirected to another page, which is specified by the `<asp:ListItem>` control's `Value` attribute. A value of `LinkButton` turns each bulleted list item into a hyperlink that posts back to the same page. It enables you to retrieve the selection that the end user makes, as illustrated in Listing 3-30.

Listing 3-30: Using the LinkButton value for the DisplayMode attribute

VB

```
<%@ Page Language="VB"%>

<script runat="server">
    Protected Sub BulletedList1_Click(ByVal sender As Object, _
       ByVal e As System.Web.UI.WebControls.BulletedListEventArgs)

         Label1.Text = "The index of item you selected: " & e.Index & _
            "<br>The value of the item selected: " & _
            BulletedList1.Items(e.Index).Text
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>BulletedList Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:BulletedList ID="BulletedList1" runat="server"
         OnClick="BulletedList1_Click" DisplayMode="LinkButton">
            <asp:ListItem>United States</asp:ListItem>
            <asp:ListItem>United Kingdom</asp:ListItem>
            <asp:ListItem>Finland</asp:ListItem>
            <asp:ListItem>Russia</asp:ListItem>
            <asp:ListItem>Sweden</asp:ListItem>
            <asp:ListItem>Estonia</asp:ListItem>
        </asp:BulletedList>
        <asp:Label ID="Label1" runat="server">
        </asp:Label>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#"%>

<script runat="server">
    protected void BulletedList1_Click(object sender,
       System.Web.UI.WebControls.BulletedListEventArgs e)
    {
         Label1.Text = "The index of item you selected: " + e.Index +
            "<br>The value of the item selected: " +
            BulletedList1.Items[e.Index].Text;
    }
</script>
```

In this example, the `DisplayMode` attribute is set to `LinkButton`, and the `OnClick` attribute is used to point to the `BulletedList1_Click` event. `BulletedList1_Click` uses the `BulletedListEventArgs` object, which only exposes the `Index` property. Using that, you can determine the index number of the item selected.

You can directly access the `Text` value of a selected item by using the `Items` property, or you can use the same property to populate an instance of the `ListItem` object, as shown here:

VB

```
Dim blSelectedValue As ListItem = BulletedList1.Items(e.Index)
```

C#

```
ListItem blSelectedValue = BulletedList1.Items[e.Index];
```

Now that you have seen how to create bulleted lists with items that you declaratively place in the code, look at how to create dynamic bulleted lists from items that are stored in a data store. The following example shows how to use the BulletedList control to data-bind to results coming from a data store; in it, all information is retrieved from an XML file.

The first step is to create the XML in Listing 3-31.

Listing 3-31: FilmChoices.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FilmChoices>
   <Film
     Title="Close Encounters of the Third Kind"
     Year="1977"
     Director="Steven Spielberg" />
   <Film
     Title="Grease"
     Year="1978"
     Director="Randal Kleiser" />
   <Film
     Title="Lawrence of Arabia"
     Year="1962"
     Director="David Lean" />
</FilmChoices>
```

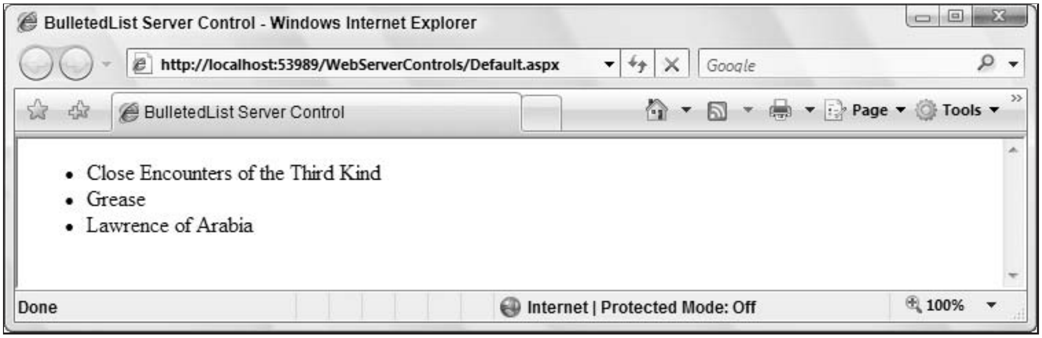
To populate the BulletedList server control with the `Title` attribute from the `FilmChoices.xml` file, use an XmlDataSource control to access the file, as illustrated in Listing 3-32.

Listing 3-32: Dynamically populating a BulletedList server control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>BulletedList Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
       <asp:BulletedList ID="BulletedList1" runat="server"
         DataSourceID="XmlDataSource1" DataTextField="Title">
       </asp:BulletedList>
       <asp:XmlDataSource ID="XmlDataSource1" runat="server"
         DataFile="~/FilmChoices.xml" XPath="FilmChoices/Film">
```

```
        </asp:XmlDataSource>
    </form>
</body>
</html>
```

In this example, you use the `DataSourceID` attribute to point to the XmlDataSource control (as you would with any control that can be bound to one of the data source controls). After you are connected to the data source control, you specifically point to the `Title` attribute using the `DataTextField` attribute. After the two server controls are connected and the page is run, you get a bulleted list that is completely generated from the contents of the XML file. Figure 3-36 shows the result.


BulletedList Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
BulletedList Server Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

- Close Encounters of the Third Kind
- Grease
- Lawrence of Arabia

Figure 3-36

The XmlDataSource server control has some limitations in that the binding to the BulletedList server control worked in the previous example only because the `Title` *value was an XML attribute and not a subelement. The XmlDataSource control exposes XML attributes as properties only when databinding. If you are going to want to work with subelements, then you are going to have to perform an XSLT transform using the XmlDataSource control's* `TransformFile` *attribute to turn elements into attributes.*

HiddenField Server Control

For many years now, developers have been using hidden fields in their Web pages to work with state management. The `<input type="hidden">` element is ideal for storing items that have no security context to them. These items are simply placeholders for data points that you want to store in the page itself instead of using the `Session` object or intermingling the data with the view state of the page. View state is another great way to store information in a page, but many developers turn off this feature to avoid corruption of the view state or possible degradation of page performance.

Any time a hidden field is placed within a Web page, it is not interpreted in the browser in any fashion, although it is completely viewable by end users if they look at the source of the HTML page.

Listing 3-33 is an example of using the HiddenField server control to hold a GUID that can be used from page to page simply by carrying over its value as the end user navigates through your application.

Listing 3-33: Working with the HiddenField server control

VB

```
<%@ Page Language="VB" %>

<script runat="server" language="vb">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
       HiddenField1.Value = System.Guid.NewGuid().ToString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>HiddenField Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
       <asp:HiddenField ID="HiddenField1" runat="Server" />
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#"%>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        HiddenField1.Value = System.Guid.NewGuid().ToString();
    }
</script>
```

In this example, the `Page_Load` event populates the `HiddenField1` control with a GUID. You can see the hidden field and its value by looking at the source of the blank HTML page that is created. You should see a result similar to the following (the GUID will have a different value, of course):

```
<input type="hidden" name="HiddenField1" id="HiddenField1"
 value="a031e77c-379b-4b4a-887c-244ee69584d5" />
```

On the page postback, ASP.NET can detect whether the HiddenField server control has changed its value since the last post. This enables you to change the HiddenField value with client-side script and then work with the changes in a page event.

The HiddenField server control has an event called `ValueChanged` that you can use when the value is changed:

VB

```
Protected Sub HiddenField1_ValueChanged(ByVal sender As Object, _
   ByVal e As System.EventArgs)
    ' Handle event here
End Sub
```

C#

```
protected void HiddenField1_ValueChanged(object sender, EventArgs e)
{
   // Handle event here
}
```

The `ValueChanged` event is triggered when the ASP.NET page is posted back to the server if the value of the HiddenField server control has changed since the last time the page was drawn. If the value has not changed, the method is never triggered. Therefore, the method is useful to act upon any changes to the HiddenField control — such as recording a value to the database or changing a value in the user's profile.

FileUpload Server Control

In ASP.NET 1.0/1.1, you could upload files using the HTML FileUpload server control. This control put an `<input type="file">` element on your Web page to enable the end user to upload files to the server. To use the file, however, you had to make a couple of modifications to the page. For example, you were required to add `enctype="multipart/form-data"` to the page's `<form>` element.

ASP.NET 2.0 introduced a new FileUpload server control that makes the process of uploading files to a server even simpler. When giving a page the capability to upload files, you simply include the new `<asp:FileUpload>` control and ASP.NET takes care of the rest, including adding the `enctype` attribute to the page's `<form>` element.

Uploading Files Using the FileUpload Control

After the file is uploaded to the server, you can also take hold of the uploaded file's properties and either display them to the end user or use these values yourself in your page's code behind. Listing 3-34 shows an example of using the new FileUpload control. The page contains a single FileUpload control, plus a Button and a Label control.

Listing 3-34: Uploading files using the new FileUpload control

VB

```
<%@ Page Language="VB"%>

<script runat="server">
   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        If FileUpload1.HasFile Then
            Try
                FileUpload1.SaveAs("C:\Uploads\" & _
                   FileUpload1.FileName)
                Label1.Text = "File name: " & _
                   FileUpload1.PostedFile.FileName & "<br>" & _
                   "File Size: " & _
                   FileUpload1.PostedFile.ContentLength & " kb<br>" & _
                   "Content type: " & _
                   FileUpload1.PostedFile.ContentType
            Catch ex As Exception
                Label1.Text = "ERROR: " & ex.Message.ToString()
            End Try
        Else
            Label1.Text = "You have not specified a file."
        End If
   End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>FileUpload Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:FileUpload ID="FileUpload1" runat="server" />
        <p>
        <asp:Button ID="Button1" runat="server" Text="Upload"
         OnClick="Button1_Click" /></p>
        <p>
        <asp:Label ID="Label1" runat="server"></asp:Label></p>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#"%>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        if (FileUpload1.HasFile)
           try {
              FileUpload1.SaveAs("C:\\Uploads\\" + FileUpload1.FileName);
              Label1.Text = "File name: " +
                   FileUpload1.PostedFile.FileName + "<br>" +
                   FileUpload1.PostedFile.ContentLength + " kb<br>" +
                   "Content type: " +
                   FileUpload1.PostedFile.ContentType;
           }
            catch (Exception ex) {
              Label1.Text = "ERROR: " + ex.Message.ToString();
            }
        else
        {
           Label1.Text = "You have not specified a file.";
        }
    }
</script>
```

From this example, you can see that the entire process is rather simple. The single button on the page initiates the upload process. The FileUpload control itself does not initiate the uploading process. You must initiate it through another event such as `Button_Click`.

When compiling and running this page, you may notice a few things in the generated source code of the page. An example of the generated source code is presented here:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1"><title>
 FileUpload Server Control
</title></head>
<body>
    <form name="form1" method="post" action="FileUpload.aspx" id="form1"
     enctype="multipart/form-data">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
 value="/wEPDwUKMTI3ODM5MzQ0Mg9kFgICAw8WAh4HZW5jdHlwZQUTbXVsdGlwYXJ0L2Zvcm
 0tZGF0YWRkrSpgAFaEKed5+5/8+zKglFfVLCE=" />
</div>

        <input type="file" name="FileUpload1" id="FileUpload1" />
        <p>
        <input type="submit" name="Button1" value="Upload" id="Button1" /></p>
        <p>
        <span id="Label1"></span></p>

<div>

<input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
  value="/wEWAgL1wLWICAKM54rGBqfR8MhZIDWVowox+TUvybG5Xj0y" />
</div></form>
</body>
</html>
```

The first thing to notice is that because the FileUpload control is on the page, ASP.NET 3.5 modified the page's `<form>` element on your behalf by adding the appropriate `enctype` attribute. Also notice that the FileUpload control was converted to an HTML `<input type="file">` element.

After the file is uploaded, the first check (done in the file's `Button1_Click` event handler) examines whether a file reference was actually placed within the `<input type="file">` element. If a file was specified, an attempt is made to upload the referenced file to the server using the `SaveAs()` method of the FileUpload control. That method takes a single `String` parameter, which should include the location where you want to save the file. In the `String` parameter used in Listing 3-34, you can see that the file is being saved to a folder called `Uploads`, which is located in the `C:\` drive.

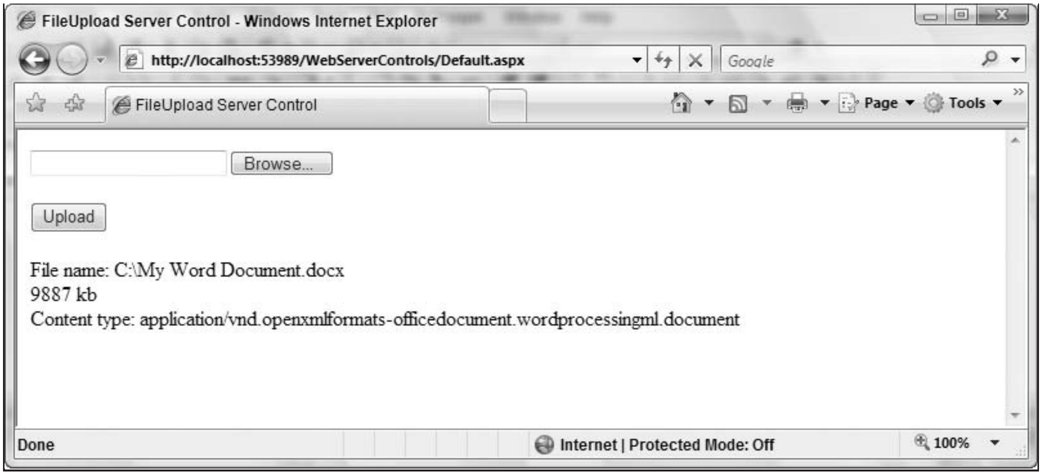
The `PostedFile.FileName` attribute is used to give the saved file the same name as the file it was copied from. If you want to name the file something else, simply use the `SaveAs()` method in the following manner:

```
FileUpload1.SaveAs("C:\Uploads\UploadedFile.txt")
```

You could also give the file a name that specifies the time it was uploaded:

```
FileUpload1.SaveAs("C:\Uploads\" & System.DateTime.Now.ToFileTimeUtc() & ".txt")
```

After the upload is successfully completed, the Label control on the page is populated with metadata of the uploaded file. In the example, the file's name, size, and content type are retrieved and displayed on the page for the end user. When the file is uploaded to the server, the page generated is similar to that shown in Figure 3-37.


FileUpload Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
FileUpload Server Control
Page
Tools
Browse...
Upload
Done
Internet | Protected Mode: Off
100%

File name: C:\My Word Document.docx
9887 kb
Content type: application/vnd.openxmlformats-officedocument.wordprocessingml.document

Figure 3-37

Uploading files to another server can be an error-prone affair. It is vital to upload files in your code using proper exception handling. That is why the file in the example is uploaded using a `Try Catch` statement.

Giving ASP.NET Proper Permissions to Upload Files

You might receive errors when your end users upload files to your Web server through the FileUpload control in your application. These might occur because the destination folder on the server is not writable for the account used by ASP.NET. If ASP.NET is not enabled to write to the folder you want, you can enable it using the folder's properties.

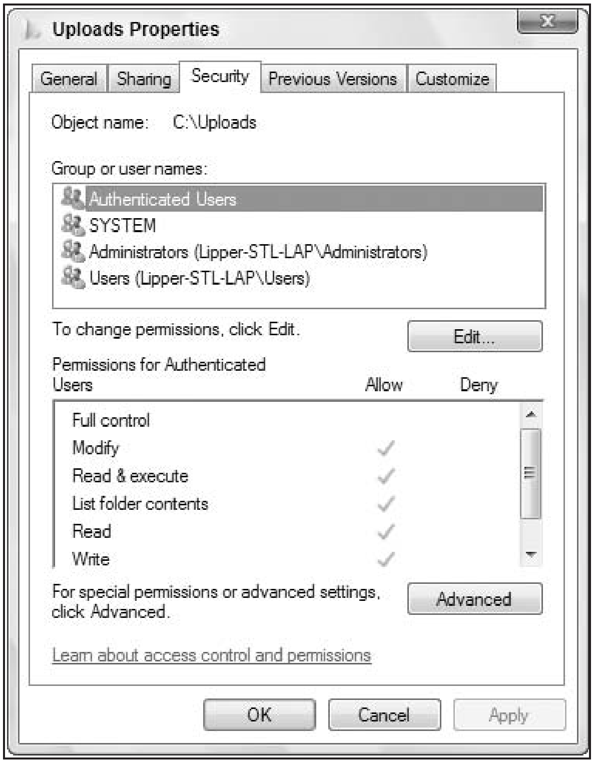
First, right-click on the folder where the ASP.NET files should be uploaded and select Properties from the provided menu. The Properties dialog for the selected folder opens. Click the Security tab to make sure the ASP.NET Machine Account is included in the list and has the proper permissions to write to disk. If it is enabled, you see something similar to what is presented in Figure 3-38.

If you do not see the ASP.NET Machine Account in the list of users allowed to access the folder, add ASP.NET by clicking the Add button and entering ASPNET (without the period) in the text area provided (see Figure 3-39).

Click OK, and you can then click the appropriate check boxes to provide the permissions needed for your application.

Understanding File Size Limitations

Your end users might never encounter an issue with the file upload process in your application, but you should be aware that some limitations exist. When users work through the process of uploading files, a size restriction is actually sent to the server for uploading. The default size limitation is 4 MB (4096 KB); the transfer fails if a user tries to upload a file that is larger than 4096 KB.


Uploads Properties
General
Sharing
Security
Previous Versions
Customize
To change permissions, click Edit.
Edit...
For special permissions or advanced settings, click Advanced.
Advanced
Learn about access control and permissions
OK
Cancel
Apply

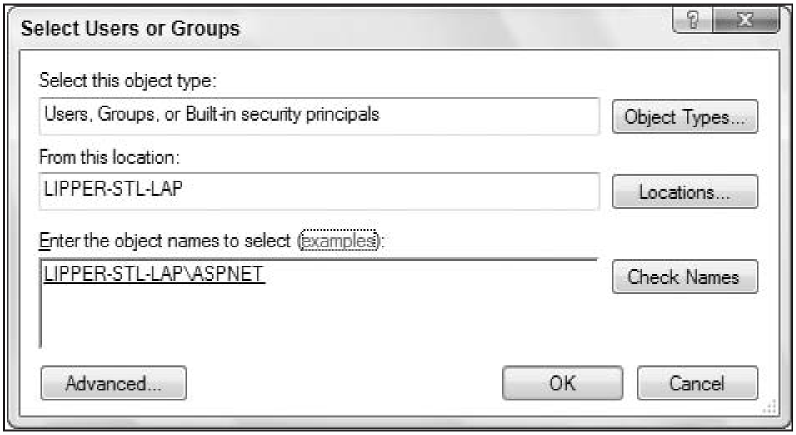
Object name: C:\Uploads

Group or user names:

- Authenticated Users
- SYSTEM
- Administrators (Lipper-STL-LAP\Administrators)
- Users (Lipper-STL-LAP\Users)

Permissions for Authenticated Users	Allow	Deny
Full control		
Modify	✓	
Read & execute	✓	
List folder contents	✓	
Read	✓	
Write	✓	

Figure 3-38


Select Users or Groups
Object Types...
Locations...
Check Names
Advanced...
OK
Cancel

Select this object type: Users, Groups, or Built-in security principals

From this location: LIPPER-STL-LAP

Enter the object names to select (examples): LIPPER-STL-LAP\ASPNET

Figure 3-39

A size restriction protects your application. You want to prevent malicious users from uploading numerous large files to your Web server in an attempt to tie up all the available processes on the server. Such an occurrence is called a *denial of service attack*. It ties up the Web server's resources so that legitimate users are denied responses from the server.

One of the great things about .NET, however, is that it usually provides a way around limitations. You can usually change the default settings that are in place. To change the limit on the allowable upload file size, you make some changes in either the root `web.config` file (found in the ASP.NET 2.0 configuration folder at `C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG`) or in your application's `web.config` file.

In the `web.config` file, you can create a node called `<httpRuntime>`. In this file, you apply the settings so that the default allowable file size is dictated by the actual request size permitted to the Web server (4096 KB). The `<httpRuntime>` section of the `web.config.comments` file is shown in Listing 3-35.

Listing 3-35: Changing the file-size limitation setting in the web.config file

```
<httpRuntime
 executionTimeout="110"
 maxRequestLength="4096"
 requestLengthDiskThreshold="80"
 useFullyQualifiedRedirectUrl="false"
 minFreeThreads="8"
 minLocalRequestFreeThreads="4"
 appRequestQueueLimit="5000"
 enableKernelOutputCache="true"
 enableVersionHeader="true"
 requireRootedSaveAsPath="true"
 enable="true"
 shutdownTimeout="90"
 delayNotificationTimeout="5"
 waitChangeNotification="0"
 maxWaitChangeNotification="0"
 enableHeaderChecking="true"
 sendCacheControlHeader="true"
 apartmentThreading="false" />
```

You can do a lot with the `<httpRuntime>` section of the `web.config` file, but two properties — the `maxRequestLength` and `executionTimeout` properties — are especially interesting.

The `maxRequestLength` property is the setting that dictates the size of the request made to the Web server. When you upload files, the file is included in the request; you alter the size allowed to be uploaded by changing the value of this property. The value presented is in kilobytes. To allow files larger than the default of 4 MB, change the `maxRequestLength` property as follows:

```
maxRequestLength="11000"
```

This example changes the `maxRequestLength` property's value to 11,000 KB (around 10 MB). With this setting in place, your end users can upload 10 MB files to the server. When changing the `maxRequestLength` property, be aware of the setting provided for the `executionTimeout` property. This property sets the time (in seconds) for a request to attempt to execute to the server before ASP.NET shuts down the request (whether or not it is finished). The default setting is `90` seconds. The end user receives a timeout error notification in the browser if the time limit is exceeded. If you are going to permit larger requests, remember that they take longer to execute than smaller ones. If you increase the size of the `maxRequestLength` property, you should examine whether to increase the `executionTimeout` property as well.

> *If you are working with smaller files, it is advisable to reduce the size allotted for the request to the Web server by decreasing the value of the* `maxRequestLength` *property. This helps safeguard your application from a denial of service attack.*

Making these changes in the `web.config` file applies this setting to all the applications that are on the server. If you want to apply this only to the application you are working with, apply the `<httpRuntime>` node to the `web.config` file of your application, overriding any setting that is in the root `web.config` file. Make sure this node resides between the `<system.web>` nodes in the configuration file.

Uploading Multiple Files from the Same Page

So far, you have seen some good examples of how to upload a file to the server without much hassle. Now, look at how to upload multiple files to the server from a single page.

No built-in capabilities in the Microsoft .NET Framework enable you to upload multiple files from a single ASP.NET page. With a little work, however, you can easily accomplish this task just as you would have in the past using .NET 1.*x*.

The trick is to import the `System.IO` class into your ASP.NET page and then to use the `HttpFileCollection` class to capture all the files that are sent in with the `Request` object. This approach enables you to upload as many files as you want from a single page.

If you wanted to, you could simply handle each and every FileUpload control on the page individually, as shown in Listing 3-36.

Listing 3-36: Handling each FileUpload control individually

VB

```
If FileUpload1.HasFile Then
   ' Handle file
End If

If FileUpload2.HasFile Then
   ' Handle file
End If
```

C#

```
if (FileUpload1.HasFile) {
   // Handle file
}

if (FileUpload2.HasFile) {
   // Handle file
}
```

If you are working with a limited number of file upload boxes, this approach works; but at the same time you may, in certain cases, want to handle the files using the `HttpFileCollection` class. This is especially true if you are working with a dynamically generated list of server controls on your ASP.NET page.

For an example of this, you can build an ASP.NET page that has three FileUpload controls and one Submit button (using the Button control). After the user clicks the Submit button and the files are posted to the server, the code behind takes the files and saves them to a specific location on the server. After the files are saved, the file information that was posted is displayed in the ASP.NET page (see Listing 3-37).

Listing 3-37: Uploading multiple files to the server

VB

```
Protected Sub Button1_Click(ByVal sender As Object, _
   ByVal e As System.EventArgs)

   Dim filepath As String = "C:\Uploads"
   Dim uploadedFiles As HttpFileCollection = Request.Files
   Dim i As Integer = 0

   Do Until i = uploadedFiles.Count
     Dim userPostedFile As HttpPostedFile = uploadedFiles(i)

     Try
        If (userPostedFile.ContentLength > 0) Then
           Label1.Text += "<u>File #" & (i + 1) & "</u><br>"
           Label1.Text += "File Content Type: " & _
              userPostedFile.ContentType & "<br>"
           Label1.Text += "File Size: " & _
              userPostedFile.ContentLength & "kb<br>"
           Label1.Text += "File Name: " & _
              userPostedFile.FileName & "<br>"

           userPostedFile.SaveAs(filepath & "\" & _
              System.IO.Path.GetFileName(userPostedFile.FileName))

           Label1.Text += "Location where saved: " & _
              filepath & "\" & _
              System.IO.Path.GetFileName(userPostedFile.FileName) & _
              "<p>"
        End If
     Catch ex As Exception
        Label1.Text += "Error:<br>" & ex.Message
     End Try
     i += 1
   Loop
End Sub
```

C#

```
protected void Button1_Click(object sender, EventArgs e)
{
   string filepath = "C:\\Uploads";
   HttpFileCollection uploadedFiles = Request.Files;

   for (int i = 0; i < uploadedFiles.Count; i++)
```

Continued

```
        {
           HttpPostedFile userPostedFile = uploadedFiles[i];

           try
           {
              if (userPostedFile.ContentLength > 0 )
              {
                 Label1.Text += "<u>File #" + (i+1) +
                    "</u><br>";
                 Label1.Text += "File Content Type: " +
                    userPostedFile.ContentType + "<br>";
                 Label1.Text += "File Size: " +
                    userPostedFile.ContentLength + "kb<br>";
                 Label1.Text += "File Name: " +
                    userPostedFile.FileName + "<br>";

                 userPostedFile.SaveAs(filepath + "\\" +
                    System.IO.Path.GetFileName(userPostedFile.FileName));

                 Label1.Text += "Location where saved: " +
                    filepath + "\\" +
                    System.IO.Path.GetFileName(userPostedFile.FileName) +
                    "<p>";
              }
           }
           catch (Exception Ex)
           {
              Label1.Text += "Error: <br>" + Ex.Message;
           }
        }
     }
```

This ASP.NET page enables the end user to select up to three files and click the Upload Files button, which initializes the `Button1_Click` event. Using the `HttpFileCollection` class with the `Request.Files` property lets you gain control over all the files that are uploaded from the page. When the files are in this state, you can do whatever you want with them. In this case, the files' properties are examined and written to the screen. In the end, the files are saved to the Uploads folder in the root directory of the server. The result of this action is illustrated in Figure 3-40.

Placing the Uploaded File into a Stream Object

One nice feature of the FileUpload control is that it not only gives you the capability to save the file to disk, but it also lets you place the contents of the file into a `Stream` object. You do this by using the `FileContent` property, as demonstrated in Listing 3-38.

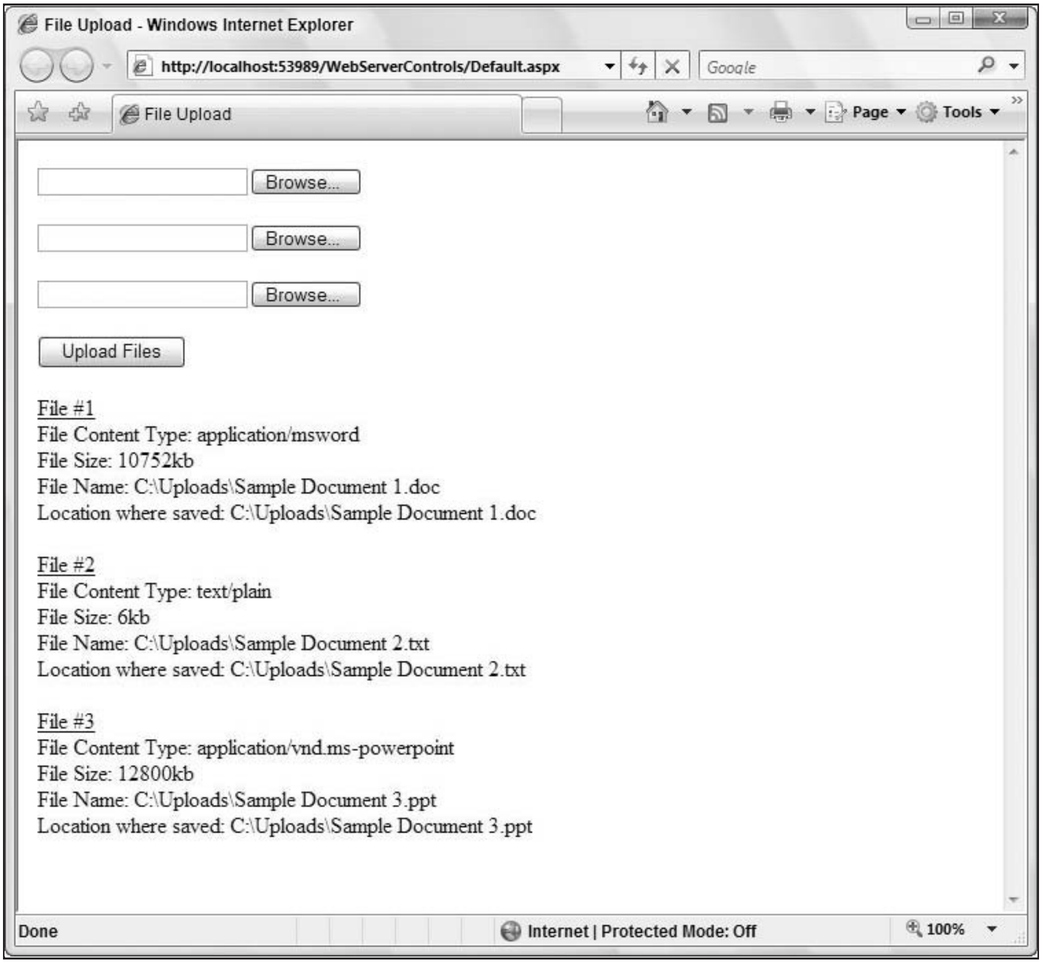
Listing 3-38: Uploading the file contents into a Stream object

VB

```
Dim myStream As System.IO.Stream
myStream = FileUpload1.FileContent
```

C#

```
System.IO.Stream myStream;
myStream = FileUpload1.FileContent;
```


File Upload - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
File Upload
Page
Tools
Browse...
Browse...
Browse...
Upload Files
Done
Internet | Protected Mode: Off
100%

File #1
File Content Type: application/msword
File Size: 10752kb
File Name: C:\Uploads\Sample Document 1.doc
Location where saved: C:\Uploads\Sample Document 1.doc

File #2
File Content Type: text/plain
File Size: 6kb
File Name: C:\Uploads\Sample Document 2.txt
Location where saved: C:\Uploads\Sample Document 2.txt

File #3
File Content Type: application/vnd.ms-powerpoint
File Size: 12800kb
File Name: C:\Uploads\Sample Document 3.ppt
Location where saved: C:\Uploads\Sample Document 3.ppt

Figure 3-40

In this short example, an instance of the `Stream` object is created. Then, using the FileUpload control's `FileContent` property, the content of the uploaded file is placed into the object. This is possible because the `FileContent` property returns a `Stream` object.

Moving File Contents from a Stream Object to a Byte Array

Because you have the capability to move the file contents to a `Stream` object of some kind, it is also fairly simple to move the contents of the file to a `Byte` array (useful for such operations as placing files in a database of some kind). To do so, first move the file contents to a `MemoryStream` object and then convert the object to the necessary `Byte` array object. Listing 3-39 shows the process.

Listing 3-39: Uploading the file contents into a Byte array

VB

```
Dim myByteArray() As Byte
Dim myStream As System.IO.MemoryStream

myStream = FileUpload1.FileContent
myByteArray = myStream.ToArray()
```

C#

```
Byte myByteArray[];
System.IO.Stream myStream;

myStream = FileUpload1.FileContent;
myByteArray = myStream.ToArray();
```

In this example, instances of a `Byte` array and a `MemoryStream` object are created. First, the `MemoryStream` object is created using the FileUpload control's `FileContent` property as you did previously. Then it's fairly simple to use the `MemoryStream` object's `ToArray()` method to populate the `myByteArray()` instance. After the file is placed into a `Byte` array, you can work with the file contents as necessary.

MultiView and View Server Controls

The MultiView and View server controls work together to give you the capability to turn on/off sections of an ASP.NET page. Turning sections on and off, which means activating or deactivating a series of View controls within a MultiView control, is similar to changing the visibility of Panel controls. For certain operations, however, you may find that the MultiView control is easier to manage and work with.

The sections, or views, do not change on the client-side; rather, they change with a postback to the server. You can put any number of elements and controls in each view, and the end user can work through the views based upon the sequence numbers that you assign to the views.

You can build these controls (like all server controls) from the source view or design view. If working with Visual Studio 2008, you can drag and drop a MultiView control onto the design surface and then drag and drop any number of View controls inside the MultiView control. Place the elements you want within the View controls. When you are finished, you have something like the view shown in Figure 3-41.

You also can create your controls directly in the code, as shown in Listing 3-40.

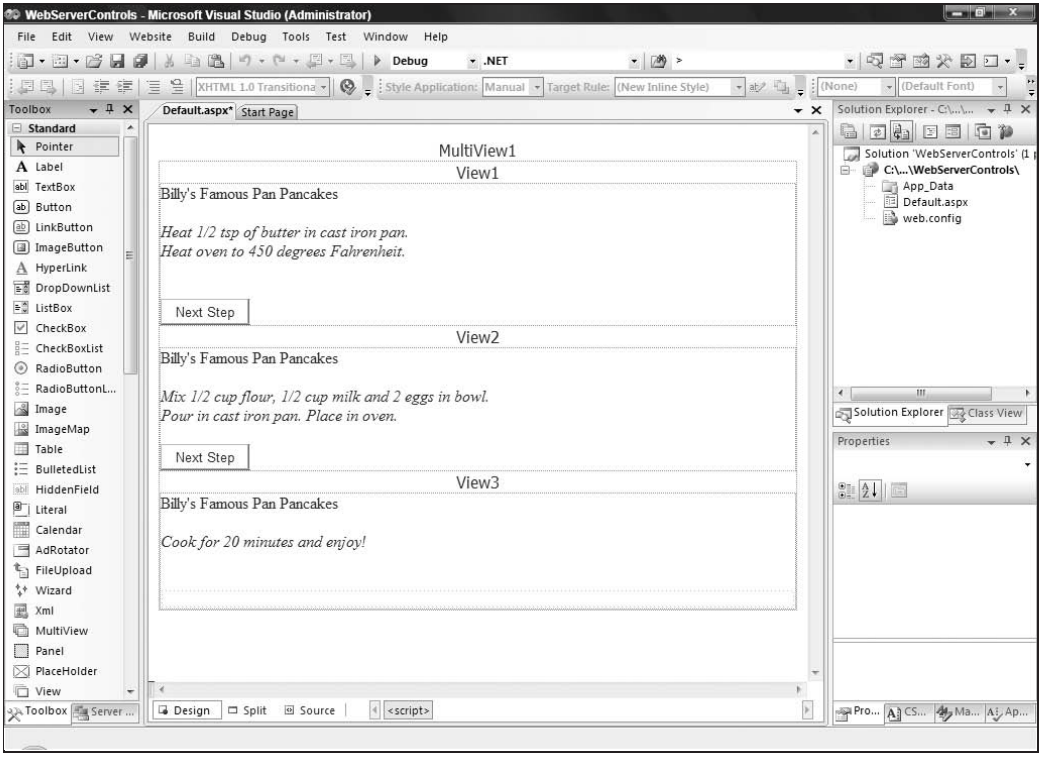
Listing 3-40: Using the MultiView and View server controls

VB

```
<%@ Page Language="VB"%>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        If Not Page.IsPostBack Then
            MultiView1.ActiveViewIndex = 0
        End If
    End Sub
    Sub NextView(ByVal sender As Object, ByVal e As System.EventArgs)
        MultiView1.ActiveViewIndex += 1
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>MultiView Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:MultiView ID="MultiView1" runat="server">
            <asp:View ID="View1" runat="server">
                Billy's Famous Pan Pancakes<p />
                <i>Heat 1/2 tsp of butter in cast iron pan.<br />
                   Heat oven to 450 degrees Fahrenheit.<br />
                </i><p />
                <asp:Button ID="Button1" runat="server" Text="Next Step"
                 OnClick="NextView" />
            </asp:View>
            <asp:View ID="View2" runat="server">
                Billy's Famous Pan Pancakes<p />
                <i>Mix 1/2 cup flour, 1/2 cup milk and 2 eggs in bowl.<br />
                   Pour in cast iron pan. Place in oven.</i><p />
                <asp:Button ID="Button2" runat="server" Text="Next Step"
                 OnClick="NextView" />
            </asp:View>
            <asp:View ID="View3" runat="server">
                Billy's Famous Pan Pancakes<p />
                <i>Cook for 20 minutes and enjoy!<br />
                </i><p />
            </asp:View>
        </asp:MultiView>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#"%>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            MultiView1.ActiveViewIndex = 0;
        }
    }

    void NextView(object sender, EventArgs e)
    {
        MultiView1.ActiveViewIndex += 1;
    }
</script>
```


WebServerControls - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Tools Test Window Help
Debug
.NET
XHTML 1.0 Transitiona
Style Application: Manual
Target Rule: (New Inline Style)
(None)
(Default Font)
Toolbox
Standard
Pointer
Label
TextBox
Button
LinkButton
ImageButton
HyperLink
DropDownList
ListBox
CheckBox
CheckBoxList
RadioButton
RadioButtonL...
Image
ImageMap
Table
BulletedList
HiddenField
Literal
Calendar
AdRotator
FileUpload
Wizard
Xml
MultiView
Panel
PlaceHolder
View
Toolbox
Server ...
Default.aspx*
Start Page
Design
Split
Source
<script>
Solution Explorer - C:\...\...
Solution 'WebServerControls' (1 p
C:\...\WebServerControls\
App_Data
Default.aspx
web.config
Solution Explorer
Class View
Properties
Pro...
CS...
Ma...
Ap...

MultiView1

View1

Billy's Famous Pan Pancakes

Heat 1/2 tsp of butter in cast iron pan.
Heat oven to 450 degrees Fahrenheit.

Next Step

View2

Billy's Famous Pan Pancakes

Mix 1/2 cup flour, 1/2 cup milk and 2 eggs in bowl.
Pour in cast iron pan. Place in oven.

Next Step

View3

Billy's Famous Pan Pancakes

Cook for 20 minutes and enjoy!

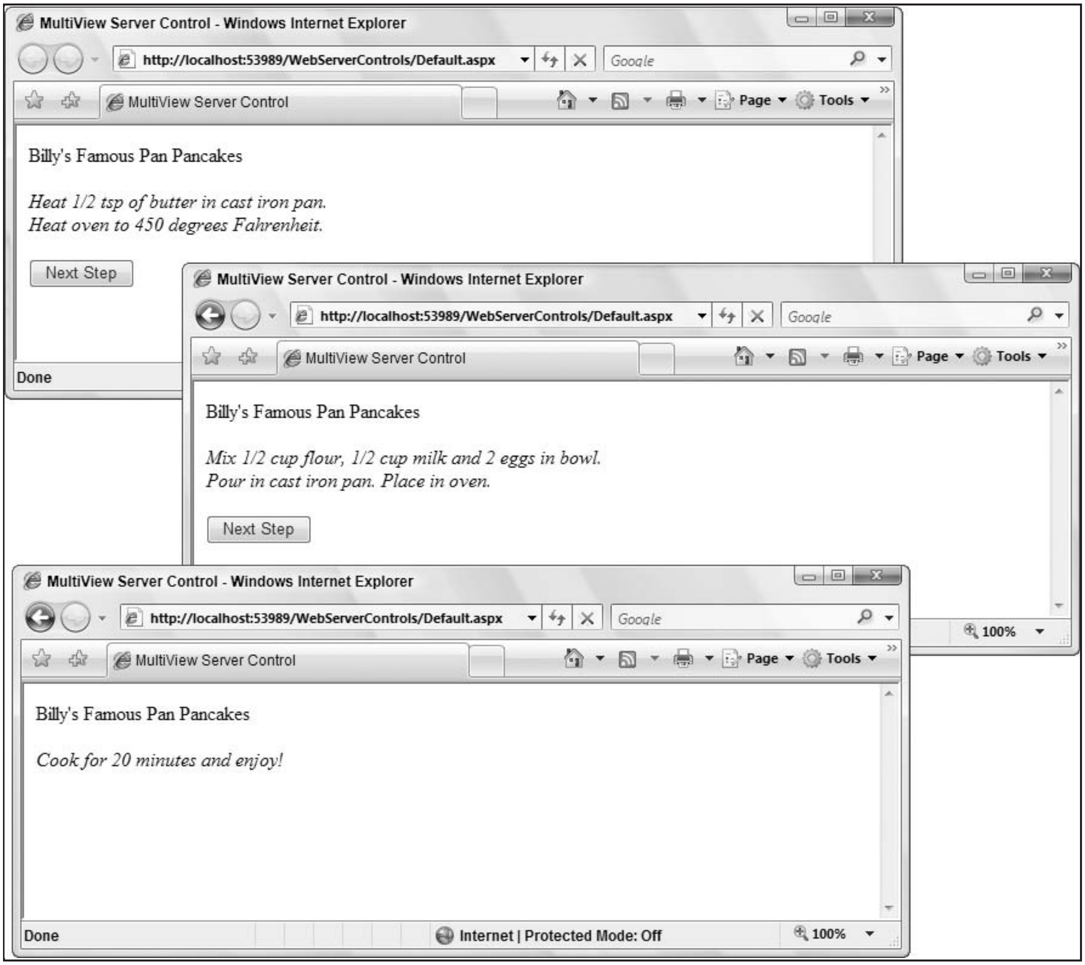
Figure 3-41

This example shows three views expressed in the MultiView control. Each view is constructed with an `<asp:View>` server control that also needs `ID` and `Runat` attributes. A button is added to each of the first two views (`View1` and `View2`) of the MultiView control. The buttons point to a server-side event that triggers the MultiView control to progress onto the next view within the series of views.

Before either of the buttons can be clicked, the MultiView control's `ActiveViewIndex` attribute is assigned a value. By default, the `ActiveViewIndex`, which describes the view that should be showing, is set to `-1`. This means that no view shows when the page is generated. To start on the first view when the page is drawn, set the `ActiveViewIndex` property to `0`, which is the first view because this is a zero-based index. Therefore, the code from Listing 3-40 first checks to see if the page is in a postback situation and if not, the `ActiveViewIndex` is assigned to the first View control.

Each of the buttons in the MultiView control triggers the `NextView` method. `NextView` simply adds one to the `ActiveViewIndex` value, thereby showing the next view in the series until the last view is shown. The view series is illustrated in Figure 3-42.

In addition to the Next Step button on the first and second views, you could place a button in the second and third views to enable the user to navigate backward through the views. To do this, create two buttons titled Previous Step in the last two views and point them to the following method in their `OnClick` events:


MultiView Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
MultiView Server Control
Page
Tools
Next Step
Done
Internet | Protected Mode: Off
100%

Billy's Famous Pan Pancakes

Heat 1/2 tsp of butter in cast iron pan.
Heat oven to 450 degrees Fahrenheit.

Billy's Famous Pan Pancakes

Mix 1/2 cup flour, 1/2 cup milk and 2 eggs in bowl.
Pour in cast iron pan. Place in oven.

Billy's Famous Pan Pancakes

Cook for 20 minutes and enjoy!

Figure 3-42

VB

```
Sub PreviousView(ByVal sender As Object, ByVal e As System.EventArgs)
    MultiView1.ActiveViewIndex -= 1
End Sub
```

C#

```
void PreviousView(object sender, EventArgs e)
{
    MultiView1.ActiveViewIndex -= 1;
}
```

Here, the `PreviousView` method subtracts one from the `ActiveViewIndex` value, thereby showing the previous view in the view series.

Another option is to spice up the MultiView control by adding a step counter that displays (to a Label control) which step in the series the end user is currently performing. In the `Page_PreRender` event, you add the following line:

VB

```
Label1.Text = "Step " & (MultiView1.ActiveViewIndex + 1).ToString() & _
   " of " & MultiView1.Views.Count.ToString()
```

C#

```
Label1.Text = "Step " + (MultiView1.ActiveViewIndex + 1).ToString() +
   " of " + MultiView1.Views.Count.ToString();
```

Now when working through the MultiView control, the end user sees `Step 1 of 3` on the first view, which changes to `Step 2 of 3` on the next view, and so on.

Wizard Server Control

Much like the MultiView control, the Wizard server control enables you to build a sequence of steps that is displayed to the end user. Web pages are all about either displaying or gathering information and, in many cases, you don't want to display all the information at once — nor do you always want to gather everything from the end user at once. Sometimes, you want to trickle the information in from or out to the end user.

When you are constructing a step-by-step process that includes logic on the steps taken, use the Wizard control to manage the entire process. The first time you use the Wizard control, notice that it allows for a far greater degree of customization than does the MultiView control.

In its simplest form, the Wizard control can be just an `<asp:Wizard>` element with any number of `<asp:WizardStep>` elements. Listing 3-41 creates a Wizard control that works through three steps.

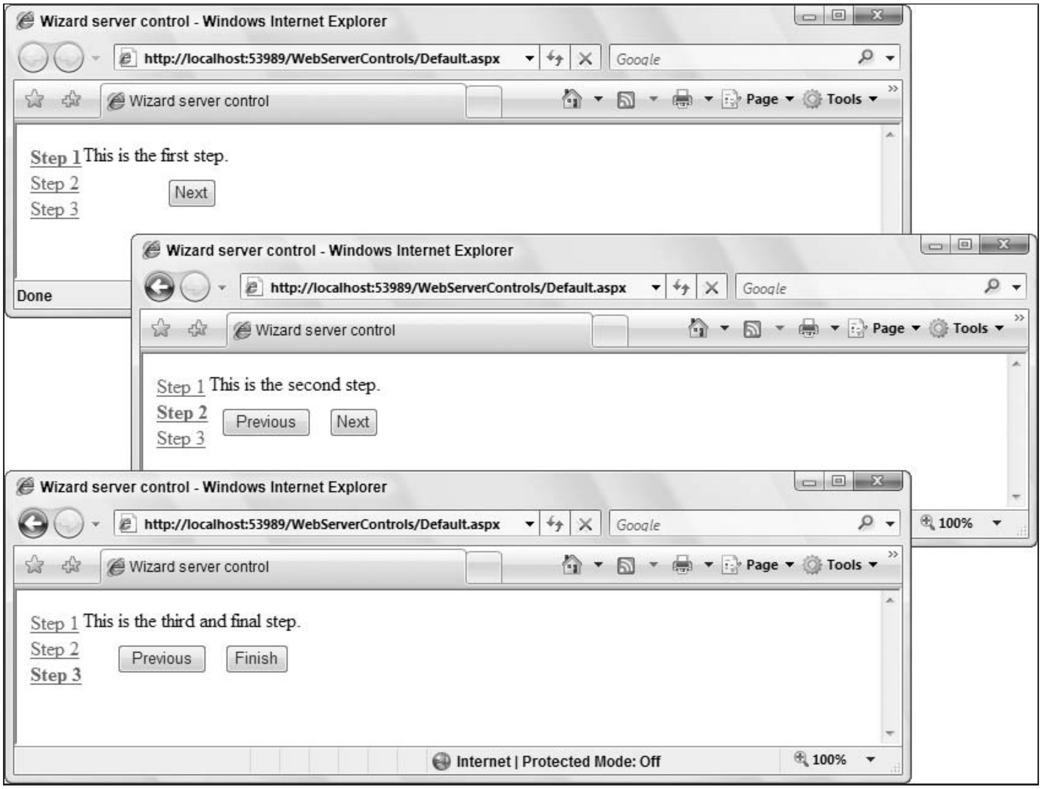
Listing 3-41: A simple Wizard control

```
<%@ Page Language="VB"%>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Wizard server control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Wizard ID="Wizard1" runat="server" DisplaySideBar="True"
         ActiveStepIndex="0">
            <WizardSteps>
                <asp:WizardStep runat="server" Title="Step 1">
                    This is the first step.</asp:WizardStep>
                <asp:WizardStep runat="server" Title="Step 2">
                    This is the second step.</asp:WizardStep>
                <asp:WizardStep runat="server" Title="Step 3">
                    This is the third and final step.</asp:WizardStep>
            </WizardSteps>
        </asp:Wizard>
    </form>
</body>
</html>
```

In this example, three steps are defined with the `<asp:WizardSteps>` control. Each step contains content — simply text in this case, although you can put in anything you want, such as other Web server controls or even user controls. The order in which the `WizardSteps` are defined is based completely on the order in which they appear within the `<WizardSteps>` element.

The `<asp:Wizard>` element itself contains a couple of important attributes. The first is `DisplaySideBar`. In this example, it is set to `True` by default — meaning that a side navigation system in the displayed control enables the end user to quickly navigate to other steps in the process. The `ActiveStepIndex` attribute of the Wizard control defines the first wizard step. In this case, it is the first step — `0`.

The three steps of the example Wizard control are shown in Figure 3-43.



This is the first step.

This is the second step.

This is the third and final step.

Wizard server control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Wizard server control
Page
Tools
Step 1
Step 2
Step 3
Next
Done
Previous
Finish
100%
Internet | Protected Mode: Off

Figure 3-43

The side navigation allows for easy access to the defined steps. The Wizard control adds appropriate buttons to the steps in the process. The first step has simply a Next button, the middle step has Previous and Next buttons, and the final step has Previous and Finish buttons. The user can navigate through the steps using either the side navigation or the buttons on each of the steps. You can customize the Wizard control in so many ways that it tends to remind me of the other rich Web server controls from ASP.NET, such as the Calendar control. Because so much is possible, only a few of the basics are covered — the ones you are most likely to employ in some of the Wizard controls you build.

Customizing the Side Navigation

The steps in the Figure 3-43 example are defined as Step 1, Step 2, and Step 3. The links are created based on the `Title` property's value that you give to each of the `<asp:WizardStep>` elements in the Wizard control:

```
<asp:WizardStep runat="server" Title="Step 1">
 This is the first step.</asp:WizardStep>
```

By default, each wizard step created in Design view is titled `Step X` (with `X` being the number in the sequence). You can easily change the value of the `Title` attributes of each of the wizard steps to define the steps as you see fit. Figure 3-44 shows the side navigation of the Wizard control with renamed titles.

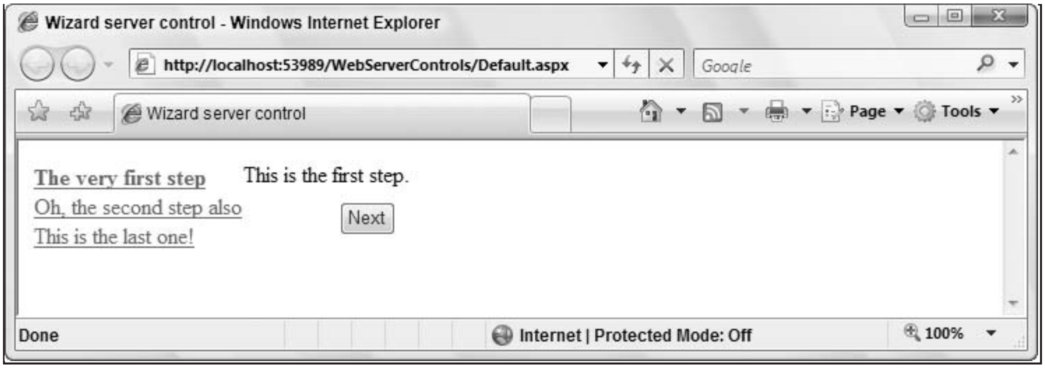

Wizard server control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Wizard server control
Page
Tools
The very first step
Oh, the second step also
This is the last one!
This is the first step.
Next
Done
Internet | Protected Mode: Off
100%

Figure 3-44

Examining the AllowReturn Attribute

Another interesting point of customization for the side navigation piece of the Wizard control is the `AllowReturn` attribute. By setting this attribute on one of the wizard steps to `False`, you can remove the capability for end users to go back to this step after they have viewed it. The end user cannot navigate backward to any viewed steps that contain the attribute, but he would be able to return to any steps that do not contain the attribute or that have it set to `True`:

```
<asp:WizardStep runat="server" Title="Step 1" AllowReturn="False">
 This is the first step.</asp:WizardStep>
```

Working with the StepType Attribute

Another interesting attribute in the `<asp:WizardStep>` element is `StepType`. The `StepType` attribute defines the structure of the buttons used on the steps. By default, the Wizard control places only a Next button on the first step. It understands that you do not need the Previous button there. It also knows to use a Next and Previous button on the middle step, and it uses Previous and Finish buttons on the last step. It draws the buttons in this fashion because, by default, the `StepType` attribute is set to `Auto`, meaning that the Wizard control determines the placement of buttons. You can, however, take control of the `StepType` attribute in the `<asp:WizardStep>` element to make your own determination about which buttons are used for which steps.

In addition to `Auto`, `StepType` value options include `Start`, `Step`, `Finish`, and `Complete`. `Start` means that the step defined has only a Next button. It simply allows the user to proceed to the next step in the series. A value of `Step` means that the wizard step has Next and Previous buttons. A value of `Finish` means that the step includes a Previous and a Finish button. `Complete` enables you to give some final message to the end user who is working through the steps of your Wizard control. In the Wizard control shown in Listing 3-42, for example, when the end user gets to the last step and clicks the Finish button, nothing happens and the user just stays on the last page. You can add a final step to give an ending message, as shown in Listing 3-42.

Listing 3-42: Having a complete step in the wizard step collection

```
<WizardSteps>
   <asp:WizardStep runat="server" Title="Step 1">
    This is the first step.</asp:WizardStep>
   <asp:WizardStep runat="server" Title="Step 2">
    This is the second step.</asp:WizardStep>
   <asp:WizardStep runat="server" Title="Step 3">
    This is the third and final step.</asp:WizardStep>
   <asp:WizardStep runat="server" Title="Final Step" StepType="Complete">
    Thanks for working through the steps.</asp:WizardStep>
</WizardSteps>
```

When you run this Wizard control in a page, you still see only the first three steps in the side navigation. Because the last step has a `StepType` set to `Complete`, it does not appear in the side navigation list. When the end user clicks the Finish button in Step 3, the last step — `Final Step` — is shown and no buttons appear with it.

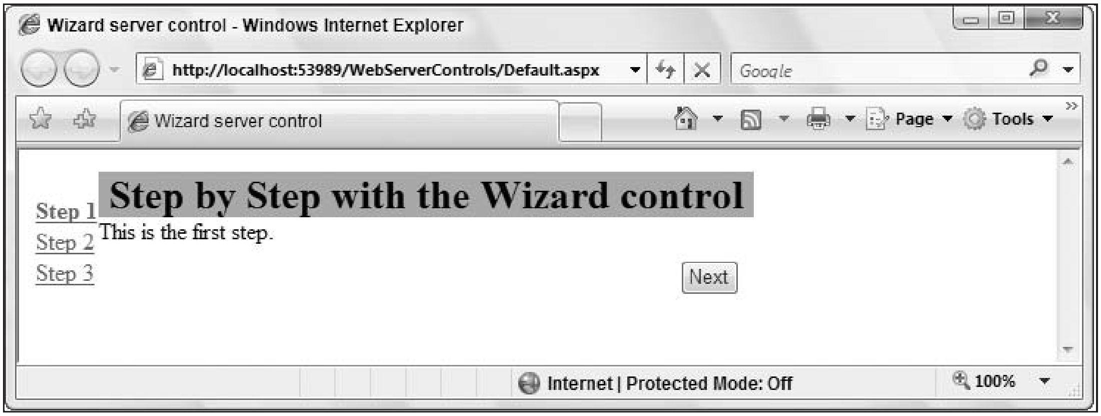
Adding a Header to the Wizard Control

The Wizard control enables you to place a header at the top of the control by means of the `HeaderText` attribute in the main `<asp:Wizard>` element. Listing 3-43 provides an example.

Listing 3-43: Working with the HeaderText attribute

```
<asp:Wizard ID="Wizard1" runat="server" ActiveStepIndex="0"
 HeaderText="&nbsp;Step by Step with the Wizard control&nbsp;"
 HeaderStyle-BackColor="DarkGray" HeaderStyle-Font-Bold="true"
 HeaderStyle-Font-Size="20">

   ...

</asp:Wizard>
```

This code creates a header that appears on each of the steps in the wizard. The result of this snippet is shown in Figure 3-45.


Wizard server control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Wizard server control
Page
Tools
Step 1
Step 2
Step 3
Next
Internet | Protected Mode: Off
100%

Step by Step with the Wizard control

This is the first step.

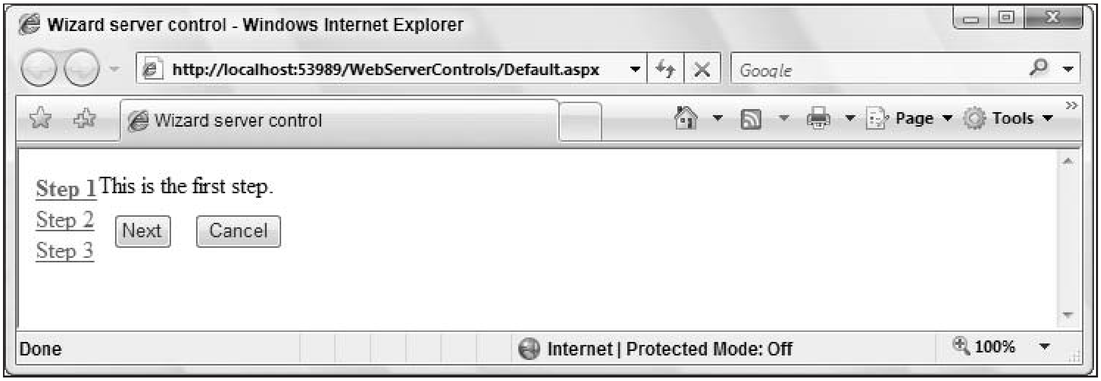
Figure 3-45

Working with the Wizard's Navigation System

As stated earlier, the Wizard control allows for a very high degree of customization — especially in the area of style. You can customize every single aspect of the process, as well as how every element appears to the end user.

Pay particular attention to the options that are available for customization of the navigation buttons. By default, the wizard steps use Next, Previous, and Finish buttons throughout the entire series of steps. From the main `<asp:Wizard>` element, you can change everything about these buttons and how they work.

First, if you look through the long list of attributes available for this element, notice that one available button is not shown by default: the Cancel button. Set the value of the `DisplayCancelButton` attribute to `True`, and a Cancel button appears within the navigation created for each and every step, including the final step in the series. Figure 3-46 shows a Cancel button in a step.


Wizard server control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Wizard server control
Page
Tools
Step 1
Step 2
Step 3
Next
Cancel
Done
Internet | Protected Mode: Off
100%

This is the first step.

Figure 3-46

After you decide which buttons to use within the Wizard navigation, you can choose their style. By default, regular buttons appear; you can change the button style with the `CancelButtonType`, `FinishStepButtonType`, `FinishStepPreviousButtonType`, `NextStepButtonType`, `PreviousStepButtonType`, and `StartStepNextButtonType` attributes. If you use any of these button types and want all the buttons consistently styled, you must change each attribute to the same value. The possible values of these button-specific elements include `Button`, `Image`, and `Link`. `Button` is the default and means that the navigation system uses buttons. A value of `Image` enables you to use image buttons, and `Link` turns a selected item in the navigation system into a hyperlink.

In addition to these button-specific attributes of the `<asp:Wizard>` element, you can also specify a URL to which the user is directed when the he clicks either the Cancel or Finish buttons. To redirect the user with one of these buttons, you use the `CancelDestinationPageUrl` or the `FinishDestinationPageUrl` attributes and set the appropriate URL as the destination.

Finally, you are not required to use the default text included with the buttons in the navigation system. You can change the text of each of the buttons with the use of the `CancelButtonText`, `FinishStepButtonText`, `FinishStepPreviousButtonText`, `NextStepButtonText`, `PreviousStepButtonText`, and the `StartStepNextButtonText` attributes.

Utilizing Wizard Control Events

One of the most convenient capabilities of the Wizard control is that it enables you to divide large forms into logical pieces. The end user can then work systematically through each section of the form. The developer, dealing with the inputted values of the form, has a few options because of the various events that are available in the Wizard control.

The Wizard control exposes events for each of the possible steps that an end user might take when working with the control. The following table describes each of the available events.

Event	**Description**
`ActiveStepChanged`	Triggers when the end user moves from one step to the next. It does not matter if the step is the middle or final step in the series. This event simply covers each step change generically.
`CancelButtonClick`	Triggers when the end user clicks the Cancel button in the navigation system.
`FinishButtonClick`	Triggers when the end user clicks the Finish button in the navigation system.
`NextButtonClick`	Triggers when the end user clicks the Next button in the navigation system.
`PreviousButtonClick`	Triggers when the end user clicks the Previous button in the navigation system.
`SideBarButtonClick`	Triggers when the end user clicks one of the links contained within the sidebar navigation of the Wizard control.

By working with these events, you can create a multi-step form that saves all the end user's input information when he changes from one step to the next. You can also use the `FinishButtonClick` event to save everything that was stored in each of the steps at the end of the process. The Wizard control remembers all the end user's input in each of the steps by means of the view state in the page, which enables you to work with all these values in the last step. It also gives the end user the capability to go back to previous steps and change values before those values are saved to a data store.

The event appears in your code behind or inline code, as shown in Listing 3-44.

Listing 3-44: The FinishButtonClick event

VB

```
<script runat="server">
    Sub Wizard1_FinishButtonClick(ByVal sender As Object, _
       ByVal e As System.Web.UI.WebControls.WizardNavigationEventArgs)

    End Sub
</script>
```

C#

```
<script runat="server">
    void Wizard1_FinishButtonClick(object sender, WizardNavigationEventArgs e)
    {

    }
</script>
```

The `OnFinishButtonClick` attribute should be added to the main `<asp:Wizard>` element to point at the new `Wizard1_FinishButtonClick` event. Listing 3-45 shows how to do this.

Listing 3-45: The <asp:Wizard> Element Changes

```
<asp:Wizard ID="Wizard1" runat="server" ActiveStepIndex="0"
 OnFinishButtonClick="Wizard1_FinishButtonClick">
```

The Wizard control is one of the great new controls that enable you to break up longer workflows into more manageable pieces for your end users. By separating longer Web forms into various wizard steps, you can effectively make your forms easy to understand and less daunting to the end user.

Using the Wizard Control to Show Form Elements

So far, you have learned how to work with each of the Wizard control steps, including how to add steps to the process and how to work with the styling of the control. Now look at how you put form elements into the Wizard control to collect information from the end user in a stepped process. This is just as simple as the first examples of the Wizard control that used only text in each of the steps.

One nice thing about putting form elements in the Wizard step process is that the Wizard control remembers each input into the form elements from step to step, enabling you to save the results of the entire form at the last step. It also means that when the end user presses the Previous button, the data that he entered into the form previously is still there and can be changed.

Work through a stepped process that enters form information by building a registration process. The last step of the process saves the results to a database of your choice, although in this example, you just push the results to a Label control on the page. Listing 3-46 shows the first part of the process.

Listing 3-46: Building the form in the Wizard control

```
<asp:Wizard ID="Wizard1" runat="Server">
   <WizardSteps>
      <asp:WizardStep ID="WizardStep1" runat="server"
       Title="Provide Personal Info">
         First name:<br />
         <asp:TextBox ID="fnameTextBox" runat="server"></asp:TextBox><br />
         Last name:<br />
         <asp:TextBox ID="lnameTextBox" runat="server"></asp:TextBox><br />
         Email:<br />
         <asp:TextBox ID="emailTextBox" runat="server"></asp:TextBox><br />
      </asp:WizardStep>
      <asp:WizardStep ID="WizardStep2" runat="server"
       Title="Membership Information">
         Are you already a member of our group?<br />
         <asp:RadioButton ID="RadioButton1" runat="server" Text="Yes"
          GroupName="Member" />
         <asp:RadioButton ID="RadioButton2" runat="server" Text="No"
          GroupName="Member" />
      </asp:WizardStep>
      <asp:WizardStep ID="WizardStep3" runat="server" Title="Provided Information"
       StepType="Complete" OnActivate="WizardStep3_Activate">
         <asp:Label ID="Label1" runat="server" />
      </asp:WizardStep>
   </WizardSteps>
</asp:Wizard>
```

This Wizard control has three steps. The first step asks for the user's personal information, and the second asks for the user's membership information. The third step contains a Label control that pushes out all the information that was input. This is done through the `Activate` event that is specific for the `WizardStep` object on the third WizardStep control. The code for the `WizardStep3_Activate` event is shown in Listing 3-47.

Listing 3-47: Adding an Activate event to a WizardStep object

VB

```
Protected Sub WizardStep3_Activate(ByVal sender As Object, _
  ByVal e As System.EventArgs)

   ' You could save the inputted data to the database here instead
   Label1.Text = "First name: " & fnameTextBox.Text.ToString() & "<br>" & _
      "Last name: " & lnameTextBox.Text.ToString() & "<br>" & _
      "Email: " & emailTextBox.Text.ToString()
End Sub
```

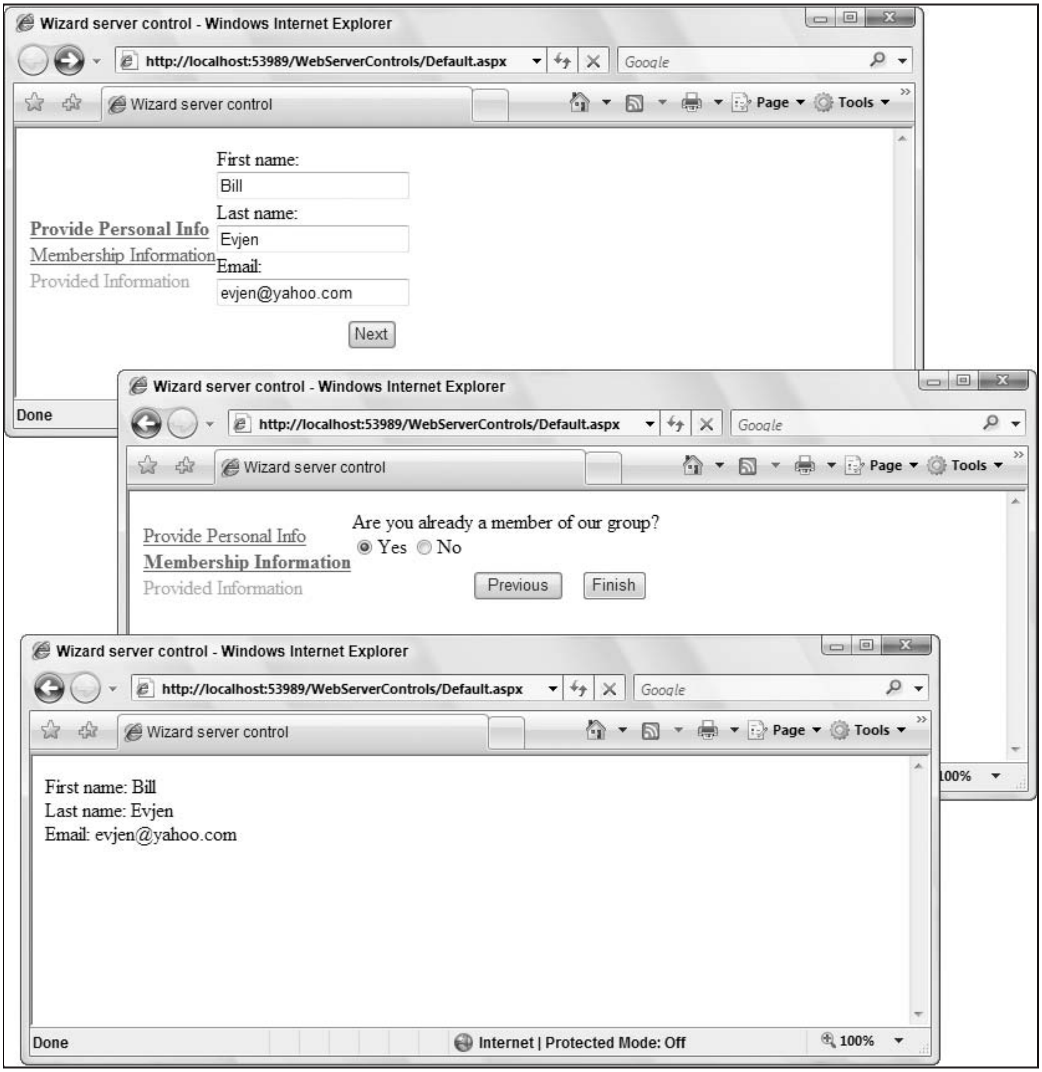
C#

```
protected void WizardStep3_Activate(object sender, EventArgs e)
```

Continued

```
{
   Label1.Text = "First name: " + fnameTextBox.Text.ToString() + "<br>" +
      "Last name: " + lnameTextBox.Text.ToString() + "<br>" +
      "Email: " + emailTextBox.Text.ToString();
}
```

When the end user comes to the third step in the display, the `WizardStep3_Activate` method from Listing 3-47 is invoked. Using the `OnActivate` attribute in the third WizardStep control, the content provided by the end user in earlier steps is used to populate a Label control. The three steps are shown in Figure 3-47.



Wizard server control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Wizard server control
Page
Tools
Provide Personal Info
Membership Information
Provided Information
Next
Done
Previous
Finish
Internet | Protected Mode: Off
100%

First name:
Bill
Last name:
Evjen
Email:
evjen@yahoo.com

Are you already a member of our group?
● Yes ○ No

First name: Bill
Last name: Evjen
Email: evjen@yahoo.com

Figure 3-47

This example is simple and straightforward, but you can increase the complexity a little bit. Imagine you want to add another WizardStep control to the process, and you want to display it only if a user specifies that he is a member in `WizardStep2`. If he answers from the radio button selection that he is not a member, you have him skip the new step and go straight to the final step where the results are displayed in the Label control. First, add an additional `WizardStep` to the Wizard control, as shown in Listing 3-48.

Listing 3-48: Adding an additional WizardStep

```
<asp:Wizard ID="Wizard1" runat="Server">
   <WizardSteps>
      <asp:WizardStep ID="WizardStep1" runat="server"
       Title="Provide Personal Info">
         First name:<br />
         <asp:TextBox ID="fnameTextBox" runat="server"></asp:TextBox><br />
         Last name:<br />
         <asp:TextBox ID="lnameTextBox" runat="server"></asp:TextBox><br />
         Email:<br />
         <asp:TextBox ID="emailTextBox" runat="server"></asp:TextBox><br />
      </asp:WizardStep>
      <asp:WizardStep ID="WizardStep2" runat="server"
       Title="Membership Information">
         Are you already a member of our group?<br />
         <asp:RadioButton ID="RadioButton1" runat="server" Text="Yes"
          GroupName="Member" />
         <asp:RadioButton ID="RadioButton2" runat="server" Text="No"
          GroupName="Member" />
      </asp:WizardStep>
      <asp:WizardStep ID="MemberStep" runat="server"
       Title="Provide Membership Number">
         Membership Number:<br />
         <asp:TextBox ID="mNumberTextBox" runat="server"></asp:TextBox>
      </asp:WizardStep>
      <asp:WizardStep ID="WizardStep3" runat="server" Title="Provided Information"
       StepType="Complete" OnActivate="WizardStep3_Activate">
         <asp:Label ID="Label1" runat="server" />
      </asp:WizardStep>
   </WizardSteps>
</asp:Wizard>
```

A single step was added to the workflow — one that simply asks the member for his membership number. Because you want to show this step only if the end user specifies that he is a member in `WizardStep2`, you add an event (shown in Listing 3-49) designed to check for that specification.

Listing 3-49: Applying logical checks on whether to show a step

VB

```
Sub Wizard1_NextButtonClick(ByVal sender As Object, _
  ByVal e As System.Web.UI.WebControls.WizardNavigationEventArgs)

   If e.NextStepIndex = 2 Then
      If RadioButton1.Checked = True Then
```

Continued

```
            Wizard1.ActiveStepIndex = 2
         Else
            Wizard1.ActiveStepIndex = 3
         End If
      End If
   End Sub
```

C#

```
void Wizard1_NextButtonClick(object sender, WizardNavigationEventArgs e)
{
   if (e.NextStepIndex == 2) {
      if (RadioButton1.Checked == true) {
         Wizard1.ActiveStepIndex = 2; }
      else {
         Wizard1.ActiveStepIndex = 3; }
   }
}
```

To check whether you should show a specific step in the process, use the `NextButtonClick` event from the Wizard control. The event uses the `WizardNavigationEventArgs` class instead of the typical `EventArgs` class that gives you access to the `NextStepIndex` number, as well as to the `CurrentStepIndex` number.

In the example from Listing 3-49, you check whether the next step to be presented in the process is `2`. Remember that this is index `2` from a zero-based index (0, 1, 2, and so on). If it is Step `2` in the index, you check which radio button is selected from the previous WizardStep. If the `RadioButton1` control is checked (meaning that the user is a member), the next step in the process is assigned as index `2`. If the `RadioButton2` control is selected, the user is not a member, and the index is then assigned as `3` (the final step), thereby bypassing the membership step in the process.

You could also take this example and alter it a bit so that you show a WizardStep only if the user is contained within a specific role (such as Admin).

Role management is covered in Chapter 16.

Showing only a WizardStep if the user is contained within a certain role is demonstrated in Listing 3-50.

Listing 3-50: Applying logical checks on whether to show a step based upon roles

VB

```
Sub Wizard1_NextButtonClick(ByVal sender As Object, _
  ByVal e As System.Web.UI.WebControls.WizardNavigationEventArgs)

   If e.NextStepIndex = 2 Then
      If (Roles.IsUserInRole("ManagerAccess") Then
         Wizard1.ActiveStepIndex = 2
      Else
         Wizard1.ActiveStepIndex = 3
      End If
   End If
End Sub
```

C#

```
void Wizard1_NextButtonClick(object sender, WizardNavigationEventArgs e)
{
   if (e.NextStepIndex == 2) {
      if (Roles.IsUserInRole("ManagerAccess")) {
         Wizard1.ActiveStepIndex = 2; }
      else {
         Wizard1.ActiveStepIndex = 3; }
      }
   }
}
```

ImageMap Server Control

The ImageMap server control enables you to turn an image into a navigation menu. In the past, many developers would break an image into multiple pieces and put it together again in a table, reassembling the pieces into one image. When the end user clicked a particular piece of the overall image, the application picked out which piece of the image was chosen and based actions upon that particular selection.

With the new ImageMap control, you can take a single image and specify particular hotspots on the image using coordinates. An example is shown in Listing 3-51.

Listing 3-51: Specifying sections of an image that are clickable

VB

```
<%@ Page Language="VB"%>

<script runat="server">
    Protected Sub Imagemap1_Click(ByVal sender As Object, _
       ByVal e As System.Web.UI.WebControls.ImageMapEventArgs)

          Response.Write("You selected: " & e.PostBackValue)
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>ImageMap Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:ImageMap ID="Imagemap1" runat="server" ImageUrl="Kids.jpg"
         Width="300" OnClick="Imagemap1_Click" HotSpotMode="PostBack">
            <asp:RectangleHotSpot Top="0" Bottom="225" Left="0" Right="150"
             AlternateText="Sofia" PostBackValue="Sofia">
            </asp:RectangleHotSpot>
            <asp:RectangleHotSpot Top="0" Bottom="225" Left="151" Right="300"
```

Continued

```
                AlternateText="Henri" PostBackValue="Henri">
               </asp:RectangleHotSpot>
           </asp:ImageMap>
       </form>
</body>
</html>
```

C#

```
<%@ page language="C#"%>

<script runat="server">
    protected void Imagemap1_Click(object sender,
       System.Web.UI.WebControls.ImageMapEventArgs e) {

          Response.Write("You selected: " + e.PostBackValue);
    }
</script>
```

This page brings up an image of my children. If you click the left side of the image, you select Sofia, and if you click the right side of the image, you select Henri. You know which child you selected through a `Response.Write` statement, as shown in Figure 3-48.

The ImageMap control enables you to specify hotspots in a couple of different ways. From the example in Listing 3-51, you can see that hotspots are placed in a rectangular fashion using the `<asp:RectangleHotSpot>` element. The control takes the `Top`, `Bottom`, `Left`, and `Right` coordinates of the rectangle that is to be the hotspot. Besides the `<asp:RectangleHotSpot>` control, you can also use the `<asp:CircleHotSpot>` and the `<asp:PolygonHotSpot>` controls. Each control takes coordinates appropriate to its shape.

After you define the hotspots on the image, you can respond to the end-user click of the hotspot in several ways. You first specify how to deal with the hotspot clicks in the root `<asp:ImageMap>` element with the use the `HotSpotMode` attribute.

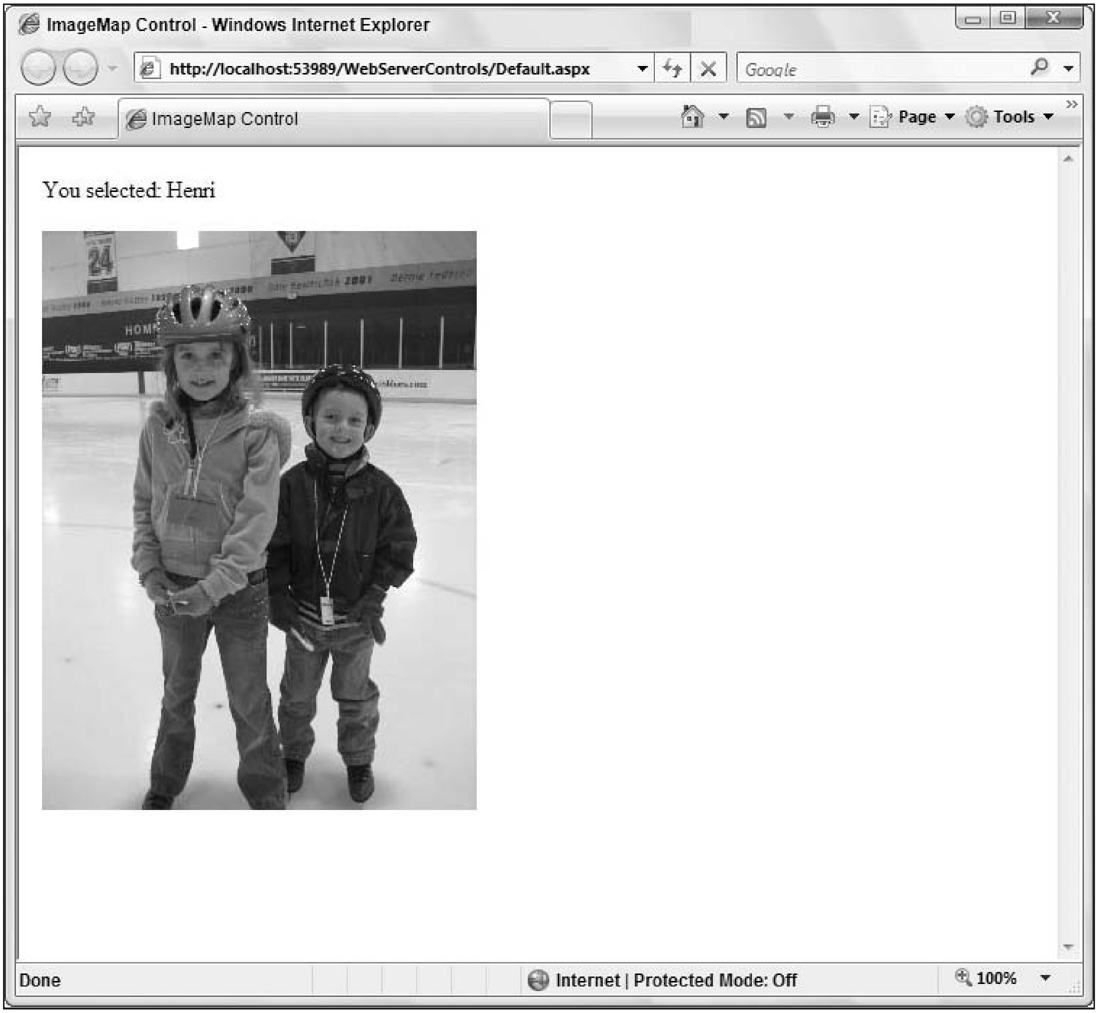
The `HotSpotMode` attribute can take the values `PostBack`, `Navigate`, or `InActive`. In the previous example, the `HotSpotMode` value is set to `PostBack` — meaning that after the end user clicks the hotspot, you want to postback to the server and deal with the click at that point.

Because the `HotSpotMode` is set to `PostBack` and you have created several hotspots, you must determine which hotspot is selected. You make this determination by giving each hotspot (`<asp:RectangleHotSpot>`) a postback value with the `PostBackValue` attribute. The example uses `Sofia` as the value of the first hotspot, and `Henri` as the value for the second.

The `PostBackValue` attribute is also the helper text that appears in the browser (in the yellow box) directly below the mouse cursor when the end user hovers the mouse over the hotspot.

After the user clicks one of the hotspots, the event procedure displays the value that was selected in a `Response.Write` statement.

Instead of posting back to the server, you can also navigate to an entirely different URL when a particular hotspot is selected. To accomplish this, change the `HotSpotMode` attribute in the main `<asp:ImageMap>` element to the value `Navigate`. Then, within the `<asp:RectangleHotSpot>` elements, simply use the


ImageMap Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
ImageMap Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

You selected: Henri

Figure 3-48

`NavigateUrl` attribute and assign the location to which the end user should be directed if that particular hotspot is clicked:

```
<asp:ImageMap ID="Imagemap1" runat="server" ImageUrl="kids.jpg"
 HotSpotMode="Navigate">
   <asp:RectangleHotSpot Top="0" Bottom="225" Left="0" Right="150"
    AlternateText="Sofia" NavigateUrl="SofiaPage.aspx">
   </asp:RectangleHotSpot>
   <asp:RectangleHotSpot Top="0" Bottom="225" Left="151" Right="300"
    AlternateText="Henri" NavigateUrl="HenriPage.aspx">
   </asp:RectangleHotSpot>
</asp:ImageMap>
```

Summary

This chapter explored numerous server controls, their capabilities, and the features they provide. With ASP.NET 3.5, you have more than 50 new server controls at your disposal.

Because you have so many server controls at your disposal when you are creating your ASP.NET applications, you have to think carefully about which is the best control for the task. Many controls seem similar, but they offer different features. These controls guarantee that you can build the best possible applications for all browsers.

Server controls are some of the most useful tools you will find in your ASP.NET arsenal. They are quite useful and can save you a lot of time. This chapter introduced you to some of these controls and to the different ways you might incorporate them into your next projects. All these controls are wonderful options to use on any of your ASP.NET pages and make it much easier to develop the functionality that your pages require.

4

Validation Server Controls

When you look at the Toolbox window in Visual Studio 2008 — especially if you've read Chapters 2 and 3, which cover the various server controls at your disposal — you may be struck by the number of server controls that come with ASP.NET 3.5. This chapter takes a look at a specific type of server control you find in the Toolbox window: the *validation server control*.

Validation server controls are a series of controls that enable you to work with the information your end users input into the form elements of the applications you build. These controls work to ensure the validity of the data being placed in the form.

Before you learn how to use these controls, however, this chapter will first take a look at the process of validation.

Understanding Validation

People have been constructing Web applications for a number of years. Usually the motivation is to provide or gather information. In this chapter, you focus on the information-gathering aspect of Web applications. If you collect data with your applications, collecting *valid* data should be important to you. If the information isn't valid, there really isn't much point in collecting it.

Validation comes in degrees. Validation is a set of rules that you apply to the data you collect. These rules can be many or few and enforced either strictly or in a lax manner: It really depends on you. No perfect validation process exists because some users may find a way to cheat to some degree, no matter what rules you establish. The trick is to find the right balance of the fewest rules and the proper strictness, without compromising the usability of the application.

The data you collect for validation comes from the Web forms you provide in your applications. Web forms are made up of different types of HTML elements that are constructed using raw HTML form elements, ASP.NET HTML server controls, or ASP.NET Web Form server controls. In the end, your forms are made up of many different types of HTML elements, such as text boxes, radio buttons, check boxes, drop-down lists, and more.

As you work through this chapter, you see the different types of validation rules that you can apply to your form elements. Remember that you have no way to validate the *truthfulness* of the information you collect; instead, you apply rules that respond to such questions as

- ❑ Is something entered in the text box?
- ❑ Is the data entered in the text box in the form of an e-mail address?

Notice from these questions that you can apply more than a single validation rule to an HTML form element (you'll see examples of this later in this chapter). In fact, you can apply as many rules to a single element as you want. Applying more rules to elements increases the strictness of the validation applied to the data.

Just remember that data collection on the Internet is one of the Internet's most important features, so you must make sure that the data you collect has value and meaning. You ensure this by eliminating any chance that the information collected does not abide by the rules you outline.

Client-Side versus Server-Side Validation

If you are new to Web application development, you might not be aware of the difference between client-side and server-side validation. Suppose that the end user clicks the Submit button on a form after filling out some information. What happens in ASP.NET is that this form is packaged in a *request* and sent to the server where the application resides. At this point in the request/response cycle, you can run validation checks on the information submitted. If you do this, it is called *server-side validation* because it occurs on the server.

On the other hand, it is also possible to supply a script (usually in the form of JavaScript) in the page that is posted to the end user's browser to perform validations on the data entered in the form *before* the form is posted back to the originating server. If this is the case, *client-side validation* has occurred.

Both types of validation have their pros and cons. Active Server Pages 2.0/3.0 developers (from the classic ASP days) are quite aware of these pros and cons because they have probably performed all the validation chores themselves. Many developers spent a considerable amount of their classic ASP programming days coding various validation techniques for performance and security.

Client-side validation is quick and responsive for the end user. It is something end users expect of the forms that they work with. If something is wrong with the form, using client-side validation ensures that the end user knows this as soon as possible. Client-side validation also pushes the processing power required of validation to the client meaning that you don't need to spin CPU cycles on the server to process the same information because the client can do the work for you.

With this said, client-side validation is the more insecure form of validation. When a page is generated in an end user's browser, this end user can look at the code of the page quite easily (simply by right-clicking his mouse in the browser and selecting View Code). When he does this, in addition to seeing the HTML code for the page, he can also see all the JavaScript that is associated with the page. If you are validating your form client-side, it doesn't take much for the crafty hacker to repost a form (containing the values he wants in it) to your server as valid. There are also the cases in which clients have simply disabled the client-scripting capabilities in their browsers — thereby making your validations useless.

Therefore, client-side validation should be considered a convenience and a courtesy to the end user and never as a security mechanism.

The more secure form of validation is server-side validation. Server-side validation means that the validation checks are performed on the server instead of on the client. It is more secure because these checks cannot be easily bypassed. Instead, the form data values are checked using server code (C# or VB) on the server. If the form isn't valid, the page is posted back to the client as invalid. Although it is more secure, server-side validation can be slow. It is sluggish simply because the page has to be posted to a remote location and checked. Your end user might not be the happiest surfer in the world if, after waiting 20 seconds for a form to post, he is told his e-mail address isn't in the correct format.

So what is the correct path? Well, actually, both! The best approach is always to perform client-side validation first and then, after the form passes and is posted to the server, to perform the validation checks again using server-side validation. This approach provides the best of both worlds. It is secure because hackers can't simply bypass the validation. They may bypass the client-side validation, but they quickly find that their form data is checked once again on the server after it is posted. This validation technique is also highly effective — giving you both the quickness and snappiness of client-side validation.

ASP.NET Validation Server Controls

In the classic ASP days, developers could spend a great deal of their time dealing with different form validation schemes. For this reason, with the initial release of ASP.NET, the ASP.NET team introduced a series of validation server controls meant to make it a snap to implement sound validation for forms.

ASP.NET not only introduces form validations as server controls, but it also makes these controls rather smart. As stated earlier, one of the tasks of classic ASP developers was to determine where to perform form validation — either on the client or on the server. The ASP.NET validation server controls eliminate this dilemma because ASP.NET performs browser detection when generating the ASP.NET page and makes decisions based on the information it gleans.

This means that if the browser can support the JavaScript that ASP.NET can send its way, the validation occurs on the client-side. If the client cannot support the JavaScript meant for client-side validation, this JavaScript is omitted and the validation occurs on the server.

The best part about this scenario is that even if client-side validation is initiated on a page, ASP.NET still performs the server-side validation when it receives the submitted page, thereby ensuring security won't be compromised. This decisive nature of the validation server controls means that you can build your ASP.NET Web pages to be the best they can possibly be — rather than dumbing-down your Web applications for the lowest common denominator.

Presently, six validation controls are available to you in ASP.NET 3.5. No new validation server controls have been added to ASP.NET since the initial release of the technology, but ASP.NET 2.0 introduced some new features, such as validation groups and new JavaScript capabilities. Both these features are discussed in this chapter. The available validation server controls include

- RequiredFieldValidator
- CompareValidator

- ❑ RangeValidator
- ❑ RegularExpressionValidator
- ❑ CustomValidator
- ❑ ValidationSummary

Working with ASP.NET validation server controls is no different from working with any other ASP.NET server control. Each of these controls allows you to drag and drop it onto a design surface or to work with it directly from the code of your ASP.NET page. These controls can also be modified so that they appear exactly as you wish — ensuring the visual uniqueness that your applications might require. You see some aspects of this throughout this chapter.

> **If the ASP.NET Validation controls don't meet your needs, you can certainly write your own custom validation controls. However, there are third-party controls available such as Peter Blum's Validation and More (VAM) from `www.peterblum.com/VAM`, which includes over 40 ASP.NET validation controls.**

The following table describes the functionality of each of the available validation server controls.

Validation Server Control	Description
RequiredFieldValidator	Ensures that the user does not skip a form entry field.
CompareValidator	Allows for comparisons between the user's input and another item using a comparison operator (equals, greater than, less than, and so on).
RangeValidator	Checks the user's input based upon a lower- and upper-level range of numbers or characters.
RegularExpressionValidator	Checks that the user's entry matches a pattern defined by a regular expression. This is a good control to use to check e-mail addresses and phone numbers.
CustomValidator	Checks the user's entry using custom-coded validation logic.
ValidationSummary	Displays all the error messages from the validators in one specific spot on the page.

Validation Causes

Validation doesn't just happen; it occurs in response to an event. In most cases, it is a button click event. The Button, LinkButton, and ImageButton server controls all have the capability to cause a page's form validation to initiate. This is the default behavior. Dragging and dropping a Button server control onto your form will give you the following initial result:

```
<asp:Button ID="Button1" Runat="server" Text="Button" />
```

If you look through the properties of the Button control, you can see that the `CausesValidation` property is set to `True`. As stated, this is the default behavior — all buttons on the page, no matter how many there are, cause the form validation to fire.

If you have multiple buttons on an ASP.NET page, and you don't want each and every button to initiate the form validation, you can set the `CausesValidation` property to `False` for all the buttons you want to ignore the validation process (for example, a form's Cancel button):

```
<asp:Button ID="Button1" Runat="server" Text="Cancel" CausesValidation="False" />
```

The RequiredFieldValidator Server Control

The RequiredFieldValidator control simply checks to see if *something* was entered into the HTML form element. It is a simple validation control, but it is one of the most frequently used. You must have a RequiredFieldValidator control for each form element on which you wish to enforce a *value-required* rule.

Listing 4-1 shows a simple use of the RequiredFieldValidator control.

Listing 4-1: A simple use of the RequiredFieldValidator server control

VB

```
<%@ Page Language="VB" %>

<script runat="server">
   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
      If Page.IsValid Then
        Label1.Text = "Page is valid!"
      End If
   End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>RequiredFieldValidator</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
        <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
         Runat="server" Text="Required!" ControlToValidate="TextBox1">
        </asp:RequiredFieldValidator>
        <br />
        <asp:Button ID="Button1" Runat="server" Text="Submit"
         OnClick="Button1_Click" />
        <br />
        <br />
        <asp:Label ID="Label1" Runat="server"></asp:Label>
```

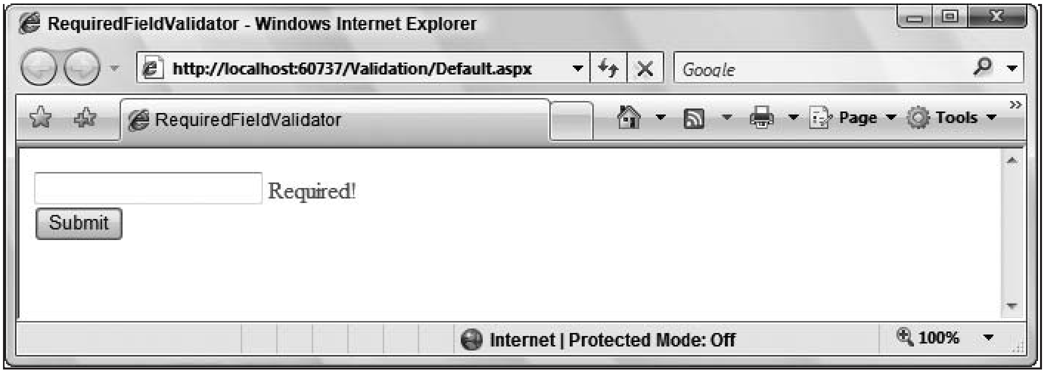
Continued

```
        </div>
        </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(Object sender, EventArgs e) {
      if (Page.IsValid) {
       Label1.Text = "Page is valid!";
      }
    }
</script>
```

Build and run this page. You are then presented with a simple text box and button on the page. Don't enter any value inside the text box, and click the Submit button. The result is shown in Figure 4-1.



RequiredFieldValidator - Windows Internet Explorer
http://localhost:60737/Validation/Default.aspx
Google
RequiredFieldValidator
Page
Tools
Submit
Internet | Protected Mode: Off
100%

Required!

Figure 4-1

Now look at the code from this example. First, nothing is different about the TextBox, Button, or Label controls. They are constructed just as they would be if you were not using any type of form validation. This page does contain a simple RequiredFieldValidator control, however. Several properties of this control are especially notable because you will use them in most of the validation server controls you create.

The first property to look at is the `Text` property. This property is the value that is shown to the end user via the Web page if the validation fails. In this case, it is a simple `Required!` string. The second property to look at is the `ControlToValidate` property. This property is used to make an association between this validation server control and the ASP.NET form element that requires the validation. In this case, the value specifies the only element in the form — the text box.

As you can see from this example, the error message is constructed from an attribute within the `<asp:RequiredFieldValidator>` control. You can also accomplish this same task by using the `Text` attribute, as shown in Listing 4-2.

Listing 4-2: Using the Text attribute

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 Runat="server" Text="Required!" ControlToValidate="TextBox1">
</asp:RequiredFieldValidator>
```

You can also express this error message between the `<asp:RequiredFieldValidator>` opening and closing nodes, as shown in Listing 4-3.

Listing 4-3: Placing values between nodes

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 Runat="server" ControlToValidate="TextBox1">
 Required!
</asp:RequiredFieldValidator>
```

Looking at the Results Generated

Again, the RequiredFieldValidator control uses client-side validation if the browser allows for such an action. You can see the client-side validation for yourself (if your browser allows for this) by right-clicking on the page and selecting View Source from the menu. In the page code, you see the JavaScript shown in Listing 4-4.

Listing 4-4: The generated JavaScript

```
... page markup removed for clarity here ...

<script type="text/javascript">
<!--
function WebForm_OnSubmit() {
if (ValidatorOnSubmit() == false) return false;
return true;
}
// -->
</script>

... page markup removed for clarity here ...

<script type="text/javascript">
<!--
var Page_Validators =  new
Array(document.getElementById("RequiredFieldValidator1"));// -->
</script>

<script type="text/javascript">
<!--
var RequiredFieldValidator1 = document.all ?
  document.all["RequiredFieldValidator1"] :
  document.getElementById("RequiredFieldValidator1");
```

Continued

```
RequiredFieldValidator1.controltovalidate = "TextBox1";
RequiredFieldValidator1.text = "Required!";
RequiredFieldValidator1.evaluationfunction =
  "RequiredFieldValidatorEvaluateIsValid";
RequiredFieldValidator1.initialvalue = "";
// -->
</script>

... page markup removed for clarity here ...

<script type="text/javascript">
<!--
var Page_ValidationActive = false;
if (typeof(ValidatorOnLoad) == "function") {
    ValidatorOnLoad();
}

function ValidatorOnSubmit() {
    if (Page_ValidationActive) {
        return ValidatorCommonOnSubmit();
    }
    else {
        return true;
    }
}
// -->
</script>
```

In the page code, you may also notice some changes to the form elements (the former server controls) that deal with the submission of the form and the associated validation requirements.

Using the InitialValue Property

Another important property when working with the RequireFieldValidator control is the `InitialValue` property. Sometimes you have form elements that are populated with some default properties (for example, from a data store), and these form elements might present the end user with values that require changes before the form can be submitted to the server.

When using the `InitialValue` property, you specify to the RequiredFieldValidator control the initial text of the element. The end user is then required to change that text value before he can submit the form. Listing 4-5 shows an example of using this property.

Listing 4-5: Working with the InitialValue property

```
<asp:TextBox ID="TextBox1" Runat="server">My Initial Value</asp:TextBox>
 &nbsp;
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 Runat="server" Text="Please change the value of the textbox!"
 ControlToValidate="TextBox1" InitialValue="My Initial Value">
</asp:RequiredFieldValidator>
```

In this case, you can see that the `InitialValue` property contains a value of `My Initial Value`. When the page is built and run, the text box contains this value as well. The RequiredFieldValidator control requires a change in this value for the page to be considered valid.

Disallowing Blank Entries and Requiring Changes at the Same Time

In the preceding example of the use of the `InitialValue` property, an interesting problem arises. First, if you run the associated example, one thing the end user can do to get past the form validation is to submit the page with no value entered in this particular text box. A blank text box does not fire a validation error because the RequiredFieldValidator control is now reconstructed to force the end user only to *change* the default value of the text box (which he did when he removed the old value). When you reconstruct the RequiredFieldValidator control in this manner, nothing in the validation rule requires that *something* be entered in the text box — just that the initial value be changed. It is possible for the user to completely bypass the form validation process by just removing anything entered in this text box.

There is a way around this, however, and it goes back to what we were saying earlier about how a form is made up of multiple validation rules — some of which are assigned to the same form element. To both require a change to the initial value of the text box and to disallow a blank entry (thereby making the element a required element), you must put an additional RequiredFieldValidator control on the page. This second RequiredFieldValidator control is associated with the same text box as the first RequiredFieldValidator control. This is illustrated in the example shown in Listing 4-6.

Listing 4-6: Using two RequiredFieldValidator controls for one form element

```
<asp:TextBox ID="TextBox1" Runat="server">My Initial Value</asp:TextBox>&nbsp;

<asp:RequiredFieldValidator ID="RequiredFieldValidator1" Runat="server"
 Text="Please change value" ControlToValidate="TextBox1"
 InitialValue="My Initial Value"></asp:RequiredFieldValidator>

<asp:RequiredFieldValidator ID="RequiredFieldValidator2" Runat="server"
 Text="Do not leave empty" ControlToValidate="TextBox1">
</asp:RequiredFieldValidator>
```

In this example, you can see that the text box does indeed have two RequiredFieldValidator controls associated with it. The first, `RequiredFieldValidator1`, requires a change to the default value of the text box through the use of the `InitialValue` property. The second RequiredFieldValidator control, `RequiredFieldValidator2`, simply makes the `TextBox1` control a form element that requires a value. You get the behavior you want by applying two validation rules to a single form element.

Validating Drop-Down Lists with the RequiredFieldValidator Control

So far, you have seen a lot of examples of using the RequiredFieldValidator control with a series of text boxes, but you can just as easily use this validation control with other form elements as well.

For example, you can use the RequiredFieldValidator control with an `<asp:DropDownList>` server control. To see this, suppose that you have a drop-down list that requires the end user to select her profession from a list of items. The first line of the drop-down list includes instructions to the end user about what to select, and you want to make this a required form element as well. The code to do this is shown in Listing 4-7.

Listing 4-7: Drop-down list validations

```
<asp:DropDownList id="DropDownList1" runat="server">
   <asp:ListItem Selected="True">Select a profession</asp:ListItem>
   <asp:ListItem>Programmer</asp:ListItem>
   <asp:ListItem>Lawyer</asp:ListItem>
   <asp:ListItem>Doctor</asp:ListItem>
   <asp:ListItem>Artist</asp:ListItem>
</asp:DropDownList>
  &nbsp;
<asp:RequiredFieldValidator id="RequiredFieldValidator1"
  runat="server" Text="Please make a selection"
  ControlToValidate="DropDownList1"
  InitialValue="Select a profession">
</asp:RequiredFieldValidator>
```

Just as when you work with the text box, the RequiredFieldValidator control in this example associates itself with the DropDownList control using the `ControlToValidate` property. The drop-down list to which the validation control is bound has an initial value — `Select a profession`. You obviously don't want your end user to retain that value when she posts the form back to the server. So again, you use the `InitialValue` property of the RequiredFieldValidator control. The value of this property is assigned to the initial selected value of the drop-down list. This forces the end user to select one of the provided professions in the drop-down list before she is able to post the form.

The CompareValidator Server Control

The CompareValidator control allows you to make comparisons between two form elements as well as to compare values contained within form elements to constants that you specify. For instance, you can specify that a form element's value must be an integer and greater than a specified number. You can also state that values must be strings, dates, or other data types that are at your disposal.

Validating Against Other Controls

One of the more common ways of using the CompareValidator control is to make a comparison between two form elements. For example, suppose that you have an application that requires users to have passwords in order to access the site. You create one text box asking for the user's password and a second text box that asks the user to confirm the password. Because the text box is in password mode, the end user cannot see what she is typing — just the number of characters that she has typed. To reduce the chances of the end user mistyping her password and inputting this incorrect password into the system, you ask her to confirm the password. After the form is input into the system, you simply have to make a comparison between the two text boxes to see if they match. If they match, it is likely that the end user typed the password correctly, and you can input the password choice into the system. If the two text boxes do not match, you want the form to be invalid. The following example, in Listing 4-8, demonstrates this situation.

Listing 4-8: Using the CompareValidator to test values against other control values

VB

```
<%@ Page Language="VB" %>
<script runat="server">
```

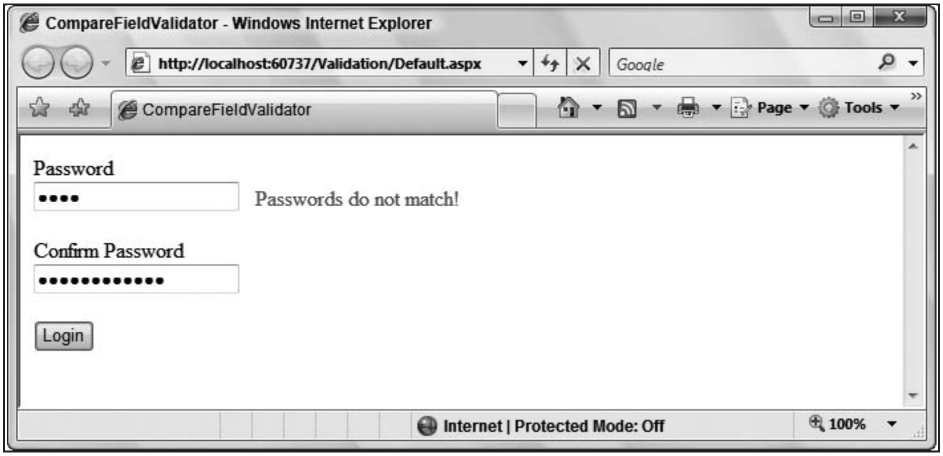
```
    Protected Sub Button1_Click(sender As Object, e As EventArgs)
       If Page.IsValid Then
          Label1.Text = "Passwords match"
       End If
    End Sub

</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CompareFieldValidator</title>
</head>
<body>
    <form runat="server" id="Form1">
        <p>
            Password<br>
            <asp:TextBox ID="TextBox1" Runat="server"
             TextMode="Password"></asp:TextBox>
             &nbsp;
            <asp:CompareValidator ID="CompareValidator1"
             Runat="server" Text="Passwords do not match!"
             ControlToValidate="TextBox2"
             ControlToCompare="TextBox1"></asp:CompareValidator>
        </p>
        <p>
            Confirm Password<br>
            <asp:TextBox ID="TextBox2" Runat="server"
             TextMode="Password"></asp:TextBox>
        </p>
        <p>
            <asp:Button ID="Button1" OnClick="Button1_Click"
             Runat="server" Text="Login"></asp:Button>
        </p>
        <p>
            <asp:Label ID="Label1" Runat="server"></asp:Label>
        </p>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>
<script runat="server">

    protected void Button1_Click(Object sender, EventArgs e) {
       if (Page.IsValid)
          Label1.Text = "Passwords match";
    }

</script>
```

Looking at the CompareValidator control on the form, you can see that it is similar to the RequiredFieldValidator control. The CompareValidator control has a property called `ControlToValidate` that associates itself with one of the form elements on the page. In this case, you need only a single CompareValidator control on the page because a single comparison is made. In this example, you are making a comparison between the value of `TextBox2` and that of `TextBox1`. Therefore, you use the `ControlToCompare` property. This specifies what value is compared to `TextBox2`. In this case, the value is `TextBox1`.

It's as simple as that. If the two text boxes do not match after the page is posted by the end user, the value of the `Text` property from the CompareValidator control is displayed in the browser. An example of this is shown in Figure 4-2.



CompareFieldValidator - Windows Internet Explorer

http://localhost:60737/Validation/Default.aspx

Password

••••

Passwords do not match!

Confirm Password

•••••••••••••

Figure 4-2

Validating Against Constants

Besides being able to validate values against values in other controls, you can also use the Compare Validator control to make comparisons against constants of specific data types. For example, suppose you have a text box on your registration form that asks for the age of the user. In most cases, you want to get back an actual number and not something such as `aa` or `bb` as a value. Listing 4-9 shows you how to ensure that you get back an actual number.

Listing 4-9: Using the CompareValidator to validate against constants

```
Age:
<asp:TextBox ID="TextBox1" Runat="server" MaxLength="3">
</asp:TextBox>
&nbsp;
<asp:CompareValidator ID="CompareValidator1" Runat="server"
  Text="You must enter a number"
  ControlToValidate="TextBox1" Type="Integer"
  Operator="DataTypeCheck"></asp:CompareValidator>
```

In this example, the end user is required to enter in a number into the text box. If she attempts to bypass the validation by entering a fake value that contains anything other than a number, the page is identified as invalid, and the CompareValidator control displays the value of the `Text` property.

To specify the data types that you want to use in these comparisons, you simply use the `Type` property. The `Type` property can take the following values:

- `Currency`
- `Date`
- `Double`
- `Integer`
- `String`

Not only can you make sure that what is entered is of a specific data type, but you can also make sure that what is entered is valid when compared to specific constants. For instance, you can make sure what is entered in a form element is greater than, less than, equal to, greater than or equal to, or less than or equal to a specified value. An example of this is illustrated in Listing 4-10.

Listing 4-10: Making comparisons with the CompareValidator control

```
Age:
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
  &nbsp;
<asp:CompareValidator ID="CompareValidator1" Runat="server"
  Operator="GreaterThan" ValueToCompare="18"
  ControlToValidate="TextBox1"
  Text="You must be older than 18 to join" Type="Integer">
</asp:CompareValidator>
```

In this case, the CompareValidator control not only associates itself with the `TextBox1` control and requires that the value must be an integer, but it also uses the `Operator` and the `ValueToCompare` properties to ensure that the number is greater than 18. Therefore, if the end user enters a value of 18 or less, the validation fails, and the page is considered invalid.

The `Operator` property can take one of the following values:

- `Equal`
- `NotEqual`
- `GreaterThan`
- `GreaterThanEqual`
- `LessThan`
- `LessThanEqual`
- `DataTypeCheck`

The `ValueToCompare` property is where you place the constant value used in the comparison. In the preceding example, it is the number `18`.

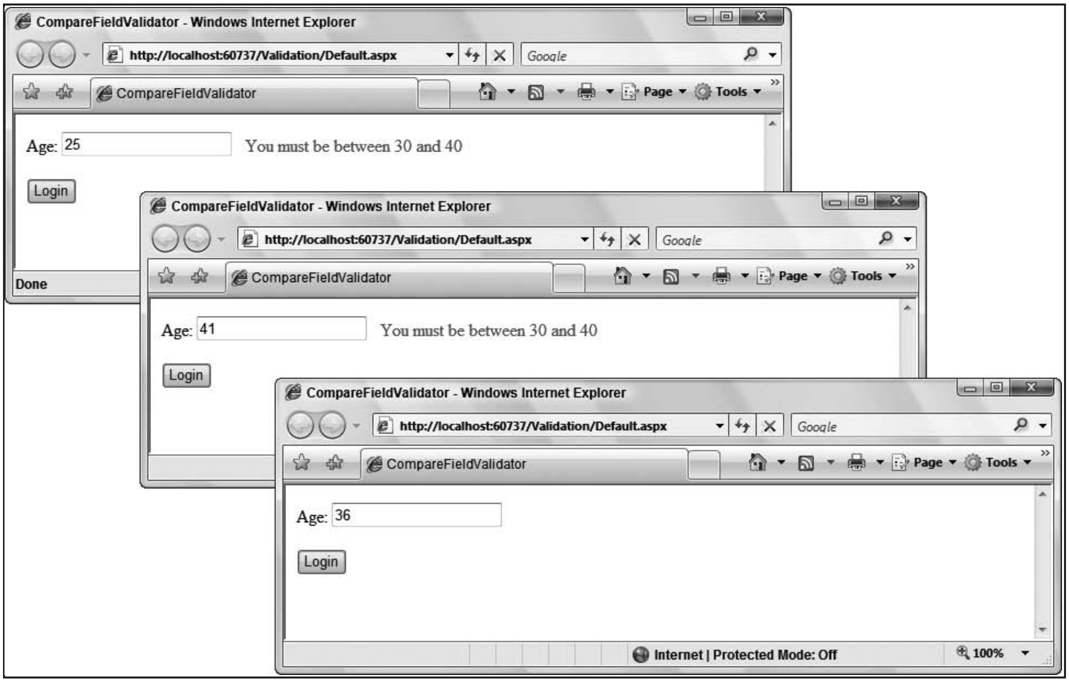
The RangeValidator Server Control

The RangeValidator control is quite similar to that of the CompareValidator control, but it makes sure that the end-user value or selection provided is between a specified range as opposed to being just greater than or less than a specified constant. For an example of this, go back to the text-box element that asks for the age of the end user and performs a validation on the value provided. This is illustrated in Listing 4-11.

Listing 4-11: Using the RangeValidator control to test an integer value

```
Age:
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
 &nbsp;
<asp:RangeValidator ID="RangeValidator1" Runat="server"
 ControlToValidate="TextBox1" Type="Integer"
 Text="You must be between 30 and 40"
 MaximumValue="40" MinimumValue="30"></asp:RangeValidator>
```

In this example, this page consists of a text box asking for the age of the end user. The RangeValidator control makes an analysis of the value provided and makes sure the value is somewhere in the range of 30 to 40. This is done using the `MaximumValue` and `MinimumValue` properties. The RangeValidator control also makes sure what is entered is an integer data type. It uses the `Type` property, which is set to `Integer`. The collection of screenshots in Figure 4-3 shows this example in action.


CompareFieldValidator - Windows Internet Explorer
http://localhost:60737/Validation/Default.aspx
Google
CompareFieldValidator
Page
Tools
Login
Done
Internet | Protected Mode: Off
100%

Age: 25 You must be between 30 and 40

Age: 41 You must be between 30 and 40

Age: 36

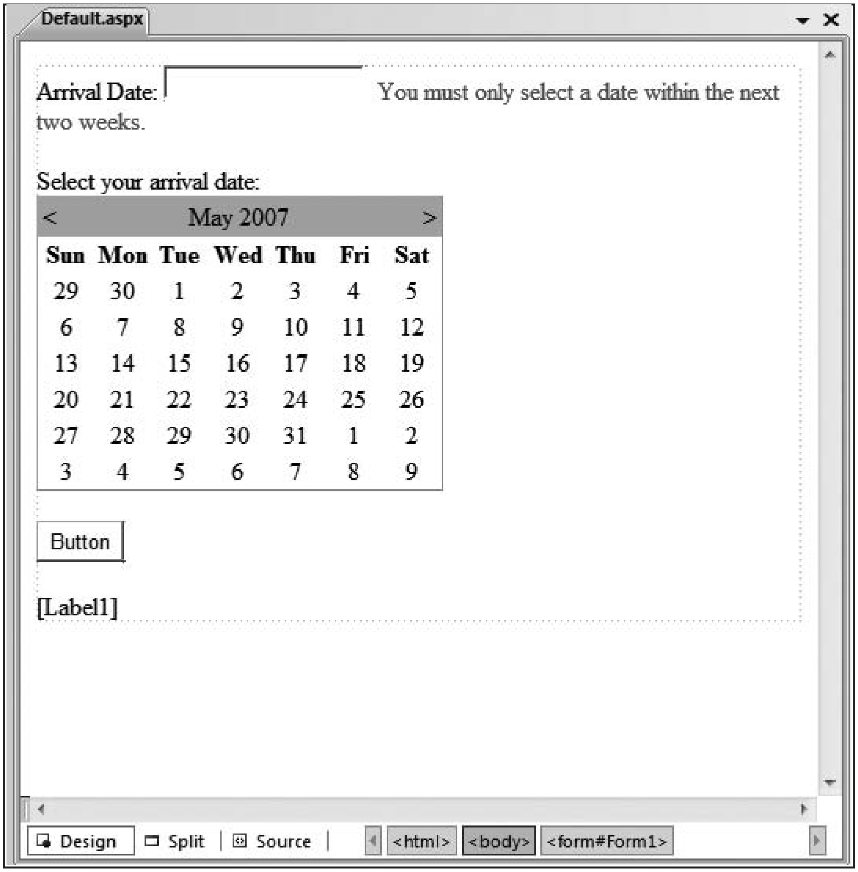
Figure 4-3

As you can see from the screenshots in Figure 4-3, a value of less than 30 causes the RangeValidator control to fire, as does a number greater than 40. A value that is somewhere between 30 and 40 (in this case 34) conforms to the validation rule of the control.

The RangeValidator control is not only about validating numbers (although it is most often used in this fashion). It can also be about validating a range of string characters as well as other items, including calendar dates. By default, the `Type` property of any of the validation controls is set to `String`. You can use the RangeValidator control to make sure what is entered in another server control (such as a calendar control) is within a certain range of dates.

For example, suppose that you are building a Web form that asks for a customer's arrival date, and the arrival date needs to be within two weeks of the current date. You can use the RangeValidator control to test for these scenarios quite easily.

Because the date range that you want to check is dynamically generated, you assign the `MaximumValue` and `MinimumValue` attribute programmatically in the `Page_Load` event. In the Designer, your sample page for this example should look like Figure 4-4.



Arrival Date: You must only select a date within the next two weeks.

Select your arrival date:

<			May 2007			>
Sun	**Mon**	**Tue**	**Wed**	**Thu**	**Fri**	**Sat**
29	30	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

Button

[Label1]

Figure 4-4

The idea is that the end user will select a date from the Calendar control, which will then populate the TextBox control. Then, when the end user clicks the form's button, he is notified if the date selected is invalid. If the date selected is valid, that date is presented through the Label control on the page. The code for this example is presented in Listing 4-12.

Listing 4-12: Using the RangeValidator control to test a string date value

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        RangeValidator1.MinimumValue = DateTime.Now.ToShortDateString()
        RangeValidator1.MaximumValue = DateTime.Now.AddDays(14).ToShortDateString()
    End Sub

    Protected Sub Calendar1_SelectionChanged(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        TextBox1.Text = Calendar1.SelectedDate.ToShortDateString()
    End Sub

    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        If Page.IsValid Then
            Label1.Text = "You are set to arrive on: " & TextBox1.Text
        End If
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Date Validation Check</title>
</head>
<body>
    <form id="form1" runat="server">
        Arrival Date:
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>&nbsp;
        <asp:RangeValidator ID="RangeValidator1" runat="server"
         Text="You must only select a date within the next two weeks."
         ControlToValidate="TextBox1" Type="Date"></asp:RangeValidator><br />
        <br />
        Select your arrival date:<br />
        <asp:Calendar ID="Calendar1" runat="server"
         OnSelectionChanged="Calendar1_SelectionChanged"></asp:Calendar>
        &nbsp;
        <br />
        <asp:Button ID="Button1" runat="server" Text="Button"
         OnClick="Button1_Click" />
        <br />
        <br />
        <asp:Label ID="Label1" runat="server"></asp:Label>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        RangeValidator1.MinimumValue = DateTime.Now.ToShortDateString();
        RangeValidator1.MaximumValue =
           DateTime.Now.AddDays(14).ToShortDateString();
    }

    protected void Calendar1_SelectionChanged(object sender, EventArgs e)
    {
        TextBox1.Text = Calendar1.SelectedDate.ToShortDateString();
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        if (Page.IsValid)
        {
            Label1.Text = "You are set to arrive on: " + TextBox1.Text.ToString();
        }
    }
</script>
```

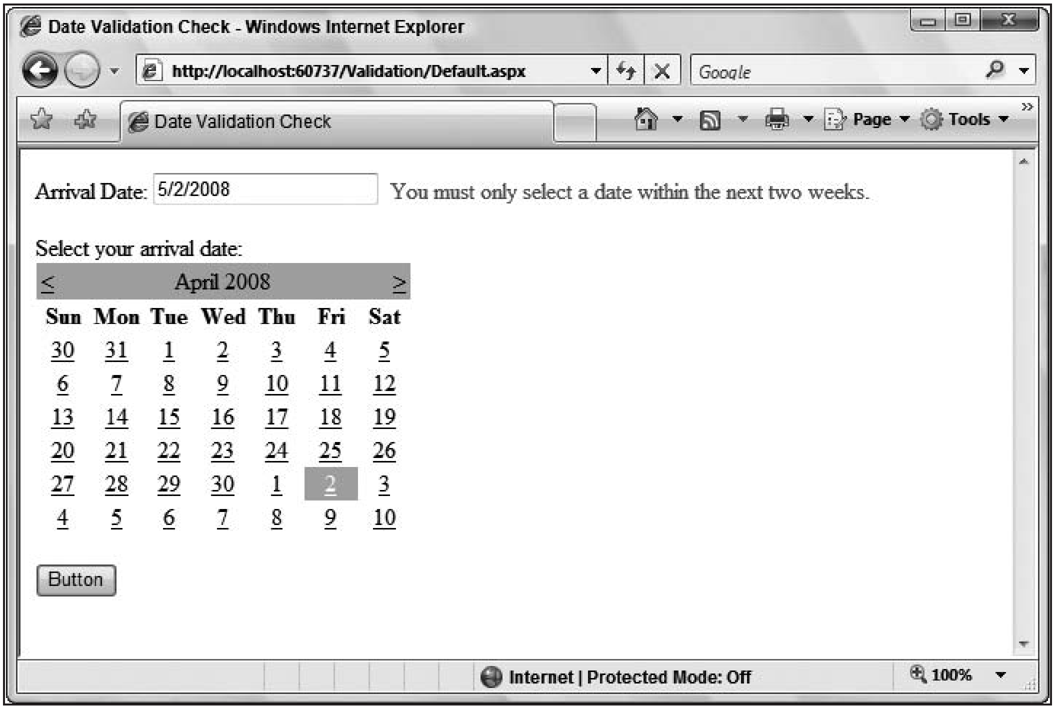
From this code, you can see that when the page is loaded, the `MinimumValue` and `MaximumValue` attributes are assigned a dynamic value. In this case, the `MinimumValue` gets the `DateTime.Now.ToShortDateString()` value, while the `MaximumValue` gets a date of 14 days later.

After the end user selects a date, the selected date is populated in the `TextBox1` control using the `Calendar1_SelectionChanged` event. After a date is selected and the button on the page is clicked, the `Button1_Click` event is fired and the page is checked for form validity using the `Page.IsValid` property. An invalid page will give you the result shown in Figure 4-5.

The RegularExpressionValidator Server Control

One exciting control that developers like to use is the RegularExpressionValidator control. This control offers a lot of flexibility when you apply validation rules to your Web forms. Using the RegularExpressionValidator control, you can check a user's input based on a pattern that you define using a regular expression.

This means that you can define a structure that a user's input will be applied against to see if its structure matches the one that you define. For instance, you can define that the structure of the user input must be in the form of an e-mail address or an Internet URL; if it doesn't match this definition, the page is considered invalid. Listing 4-13 shows you how to validate what is input into a text box by making sure it is in the form of an e-mail address.


Date Validation Check - Windows Internet Explorer
http://localhost:60737/Validation/Default.aspx
Google
Date Validation Check
Page
Tools
Internet | Protected Mode: Off
100%

Arrival Date: 5/2/2008 You must only select a date within the next two weeks.

Select your arrival date:

<			April 2008			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	1	2	3
4	5	6	7	8	9	10

Button

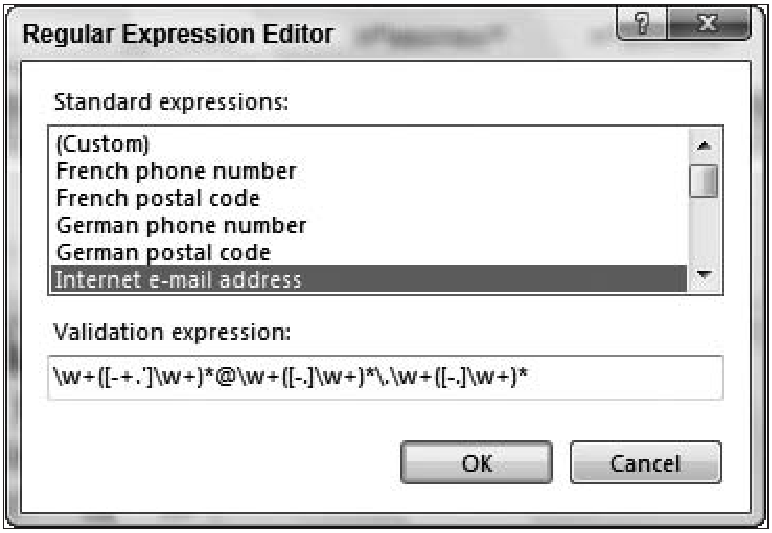
Figure 4-5

Listing 4-13: Making sure the text-box value is an e-mail address

```
Email:
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
  &nbsp;
<asp:RegularExpressionValidator ID="RegularExpressionValidator1"
  Runat="server" ControlToValidate="TextBox1"
  Text="You must enter an email address"
  ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
</asp:RegularExpressionValidator>
```

Just like the other validation server controls, the RegularExpressionValidator control uses the `ControlToValidate` property to bind itself to the TextBox control, and it includes a `Text` property to push out the error message to the screen if the validation test fails. The unique property of this validation control is the `ValidationExpression` property. This property takes a string value, which is the regular expression you are going to apply to the input value.

Visual Studio 2008 makes it a little easier to use regular expressions through the use of the Regular Expression Editor. This editor provides a few commonly used regular expressions that you might want to apply to your RegularExpressionValidator. To get at this editor, you work with your page from Design view. Be sure to highlight the `RegularExpressionValidator1` server control in this Design view to see the control's properties. In the Property window of Visual Studio, click the button found next to the `ValidationExpression` property to launch the Regular Expression Editor. This editor is shown in Figure 4-6.



Standard expressions: Internet e-mail address

Validation expression: \w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*

Figure 4-6

Using this editor, you can find regular expressions for things like e-mail addresses, Internet URLs, zip codes, phone numbers, and social security numbers. In addition to working with the Regular Expression Editor to help you with these sometimes complicated regular expression strings, you can also find a good-sized collection of them at an Internet site called RegExLib found at `www.regexlib.com`.

The CustomValidator Server Control

So far, you have seen a wide variety of validation controls that are at your disposal. In many cases, these validation controls address many of the validation rules that you want to apply to your Web forms. Sometimes, however, none of these controls works for you, and you have to go beyond what they offer. This is where the CustomValidator control comes into play.

The CustomValidator control allows you to build your own client-side or server-side validations that can then be easily applied to your Web forms. Doing so allows you to make validation checks against values or calculations performed in the data tier (for example, in a database), or to make sure that the user's input validates against some arithmetic validation (for example, determining if a number is even or odd). You can do quite a bit with the CustomValidator control.

Using Client-Side Validation

One of the worthwhile functions of the CustomValidator control is its capability to easily provide custom client-side validations. Many developers have their own collections of JavaScript functions they employ in their applications, and using the CustomValidator control is one easy way of getting these functions implemented.

For example, look at a simple form that asks for a number from the end user. This form uses the CustomValidator control to perform a custom client-side validation on the user input to make sure that the number provided is divisible by 5. This is illustrated in Listing 4-14.

Listing 4-14: Using the CustomValidator control to perform client-side validations

VB

```
<%@ Page Language="VB" %>

<script runat="server">
   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
       Label1.Text = "VALID NUMBER!"
   End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CustomValidator</title>

    <script type="text/javascript">
        function validateNumber(oSrc, args) {
           args.IsValid = (args.Value % 5 == 0);
        }
    </script>

</head>
<body>
    <form id="form1" runat="server">
    <div>
        <p>
            Number:
            <asp:TextBox ID="TextBox1"
             Runat="server"></asp:TextBox>
             &nbsp;
            <asp:CustomValidator ID="CustomValidator1"
             Runat="server" ControlToValidate="TextBox1"
             Text="Number must be divisible by 5"
             ClientValidationFunction="validateNumber">
            </asp:CustomValidator>
        </p>
        <p>
            <asp:Button ID="Button1" OnClick="Button1_Click"
             Runat="server" Text="Button"></asp:Button>
        </p>
        <p>
            <asp:Label ID="Label1" Runat="server"></asp:Label>
        </p>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>
<script runat="server">

    protected void Button1_Click(Object sender, EventArgs e) {
       Label1.Text = "VALID NUMBER!";
    }

</script>
```

Looking over this Web form, you can see a couple of things happening. First, it is a simple form with only a single text box requiring user input. The user clicks the button that triggers the `Button1_Click` event, which in turn populates the `Label1` control on the page. It carries out this simple operation only if all the validation checks are performed and the user input passes these tests.

One item that is different about this page is the inclusion of the second `<script>` block found within the `<head>` section. This is the custom JavaScript. Note that Visual Studio 2008 is very friendly toward these kinds of constructions, even when you are switching between the Design and Code views of the page — something previous Visual Studio editions were rather poor at dealing with. This JavaScript function — `validateNumber` — is shown here:

```
<script type="text/javascript">
   function validateNumber(oSrc, args) {
      args.IsValid = (args.Value % 5 == 0);
    }
</script>
```

This second `<script>` section is the client-side JavaScript that you want the CustomValidator control to use when making its validation checks on the information entered into the text box. The JavaScript functions you employ are going to use the `args.IsValid` property and set this property to either `true` or `false` depending on the outcome of the validation check. In this case, the user input (`args.Value`) is checked to see if it is divisible by `5`. The Boolean value returned is then assigned to the `args.IsValid` property, which is then used by the CustomValidator control.

The CustomValidator control, like the other controls before it, uses the `ControlToValidate` property to associate itself with a particular element on the page. The property that you are interested in here is the `ClientValidationFunction` property. The string value provided to this property is the name of the client-side function that you want this validation check to employ when the CustomValidator control is triggered. In this case, it is `validateNumber`:

```
ClientValidationFunction="validateNumber"
```

If you run this page and make an invalid entry, you produce the result illustrated in Figure 4-7.

Using Server-Side Validation

Now let's move this same validation check from the client to the server. The CustomValidator control allows you to make custom server-side validations a reality as well. You will find that creating your server-side validations is just as easy as creating client-side validations.

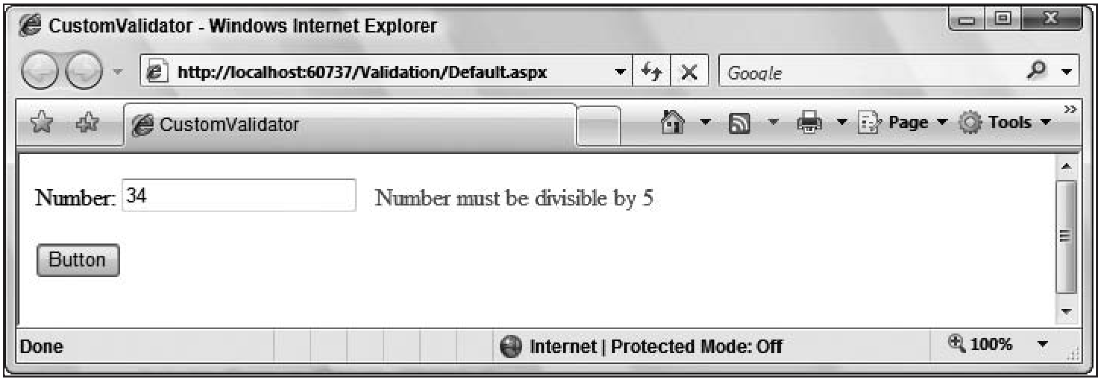

CustomValidator - Windows Internet Explorer
http://localhost:60737/Validation/Default.aspx
Google
CustomValidator
Page
Tools
Number:
34
Number must be divisible by 5
Button
Done
Internet | Protected Mode: Off
100%

Figure 4-7

If you create your own server-side validations, you can make them as complex as your applications require. For instance, using the CustomValidator for server-side validations is something you do if you want to check the user's input against dynamic values coming from XML files, databases, or elsewhere.

For an example of using the CustomValidator control for some custom server-side validation, you can work with the same example as you did when creating the client-side validation. Now, create a server-side check that makes sure a user-input number is divisible by 5. This is illustrated in Listing 4-15.

Listing 4-15: Using the CustomValidator control to perform server-side validations

VB

```
<%@ Page Language="VB" %>

<script runat="server">
   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
       If Page.IsValid Then
          Label1.Text = "VALID ENTRY!"
       End If
   End Sub

   Sub ValidateNumber(sender As Object, args As ServerValidateEventArgs)
       Try
          Dim num As Integer = Integer.Parse(args.Value)
          args.IsValid = ((num mod 5) = 0)
       Catch ex As Exception
          args.IsValid = False
       End Try
   End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CustomValidator</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <p>
            Number:
            <asp:TextBox ID="TextBox1"
             Runat="server"></asp:TextBox>
             &nbsp;
            <asp:CustomValidator ID="CustomValidator1"
             Runat="server" ControlToValidate="TextBox1"
             Text="Number must be divisible by 5"
             OnServerValidate="ValidateNumber"></asp:CustomValidator>
        </p>
        <p>
            <asp:Button ID="Button1" OnClick="Button1_Click"
             Runat="server" Text="Button"></asp:Button>
        </p>
        <p>
            <asp:Label ID="Label1" Runat="server"></asp:Label>
        </p>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">

    protected void Button1_Click(Object sender, EventArgs e) {
         if (Page.IsValid) {
            Label1.Text = "VALID ENTRY!";
         }
    }

    void ValidateNumber(object source, ServerValidateEventArgs args)
    {
       try
       {
          int num = int.Parse(args.Value);
          args.IsValid = ((num%5) == 0);
       }
       catch(Exception ex)
       {
          args.IsValid = false;
       }
    }

</script>
```

Instead of a client-side JavaScript function in the code, this example includes a server-side function — `ValidateNumber`. The `ValidateNumber` function, as well as all functions that are being constructed to work with the CustomValidator control, must use the `ServerValidateEventArgs` object as one of the parameters in order to get the data passed to the function for the validation check. The `ValidateNumber` function itself is nothing fancy. It simply checks to see if the provided number is divisible by 5.

From within your custom function, which is designed to work with the CustomValidator control, you actually get at the value coming from the form element through the `args.Value` object. Then you set the `args.IsValid` property to either `True` or `False` depending on your validation checks. From the preceding example, you can see that the `args.IsValid` is set to `False` if the number is not divisible by 5 and also that an exception is thrown (which would occur if a string value was input into the form element). After the custom function is established, the next step is to apply it to the CustomValidator control, as shown in the following example:

```
<asp:CustomValidator ID="CustomValidator1"
 Runat="server" ControlToValidate="TextBox1"
 Text="Number must be divisible by 5"
 OnServerValidate="ValidateNumber"></asp:CustomValidator>
```

To make the association between a CustomValidator control and a function that you have in your server-side code, you simply use the `OnServerValidate` attribute. The value assigned to this property is the name of the function — in this case, `ValidateNumber`.

Running this example causes the postback to come back to the server and the validation check (based on the `ValidateNumber` function) to be performed. From here, the page reloads and the `Page_Load` event is called. In the example from Listing 4-15, you can see that a check is done to see whether the page is valid. This is done using the `Page.IsValid` property:

```
If Page.IsValid Then
   Label1.Text = "VALID ENTRY!"
End If
```

Using Client-Side and Server-Side Validation Together

As stated earlier in this chapter, you have to think about the security of your forms and to ensure that the data you are collecting from the forms is valid data. For this reason, when you decide to employ client-side validations (as you did in Listing 4-14), you should take steps to also reconstruct the client-side function as a server-side function. When you have done this, you should associate the CustomValidator control to both the client-side and server-side functions. In the case of the number check validation from Listings 4-14 and 4-15, you can use both validation functions in your page and then change the CustomValidator control to point to both of these functions, as shown in Listing 4-16.

Listing 4-16: The CustomValidator control with client- and server-side validations

```
<asp:CustomValidator ID="CustomValidator1"
    Runat="server" ControlToValidate="TextBox1"
    Text="Number must be divisible by 5"
    ClientValidationFunction="validateNumber"
    OnServerValidate="ValidateNumber"></asp:CustomValidator>
```

From this example, you can see it is simply a matter of using both the `ClientValidationFunction` and the `OnServerValidate` attributes at the same time.

The ValidationSummary Server Control

The ValidationSummary control is not a control that performs validations on the content input into your Web forms. Instead, this control is the reporting control, which is used by the other validation controls on a page. You can use this validation control to consolidate error reporting for all the validation errors that occur on a page instead of leaving this up to each and every individual validation control.

You might want this capability for larger forms, which have a comprehensive form validation process. If this is the case, you may find it rather user-friendly to have all the possible validation errors reported to the end user in a single and easily identifiable manner. These error messages can be displayed in a list, bulleted list, or paragraph.

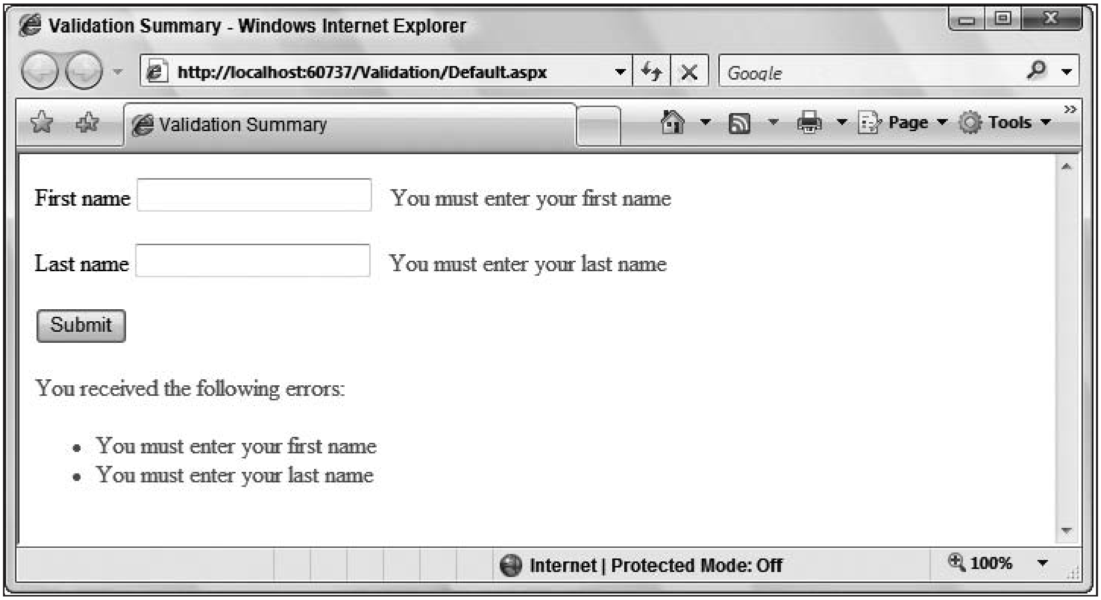
By default, the ValidationSummary control shows the list of validation errors as a bulleted list. This is illustrated in Listing 4-17.

Listing 4-17: A partial page example of the ValidationSummary control

```
<p>First name
   <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
    &nbsp;
   <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
    Runat="server" ErrorMessage="You must enter your first name"
    ControlToValidate="TextBox1"></asp:RequiredFieldValidator>
</p>
<p>Last name
   <asp:TextBox ID="TextBox2" Runat="server"></asp:TextBox>
    &nbsp;
   <asp:RequiredFieldValidator ID="RequiredFieldValidator2"
    Runat="server" ErrorMessage="You must enter your last name"
    ControlToValidate="TextBox2"></asp:RequiredFieldValidator>
</p>
<p>
   <asp:Button ID="Button1" OnClick="Button1_Click" Runat="server"
    Text="Submit"></asp:Button>
</p>
<p>
   <asp:ValidationSummary ID="ValidationSummary1" Runat="server"
    HeaderText="You received the following errors:">
   </asp:ValidationSummary>
</p>
<p>
   <asp:Label ID="Label1" Runat="server"></asp:Label>
</p>
```

This example asks the end user for her first and last name. Each text box in the form has an associated RequiredFieldValidator control assigned to it. When the page is built and run, if the user clicks the Submit button with no values placed in either of the text boxes, it causes both validation errors to fire. This result is shown in Figure 4-8.

As in earlier examples of validation controls on the form, these validation errors appear next to each of the text boxes. You can see, however, that the ValidationSummary control also displays the validation errors as a bulleted list in red at the location of the control on the Web form. In most cases, you do not want these errors to appear twice on a page for the end user. You can change this behavior by using the `Text` property of the validation controls, in addition to the `ErrorMessage` property, as you have typically done throughout this chapter. This approach is shown in Listing 4-18.


Validation Summary - Windows Internet Explorer
http://localhost:60737/Validation/Default.aspx
Google
Validation Summary
Page
Tools
First name
Last name
Submit
Internet | Protected Mode: Off
100%

You must enter your first name

You must enter your last name

You received the following errors:

- You must enter your first name
- You must enter your last name

Figure 4-8

Listing 4-18: Using the Text property of a validation control

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 Runat="server" ErrorMessage="You must enter your first name" Text="*"
 ControlToValidate="TextBox1"></asp:RequiredFieldValidator>
```

or

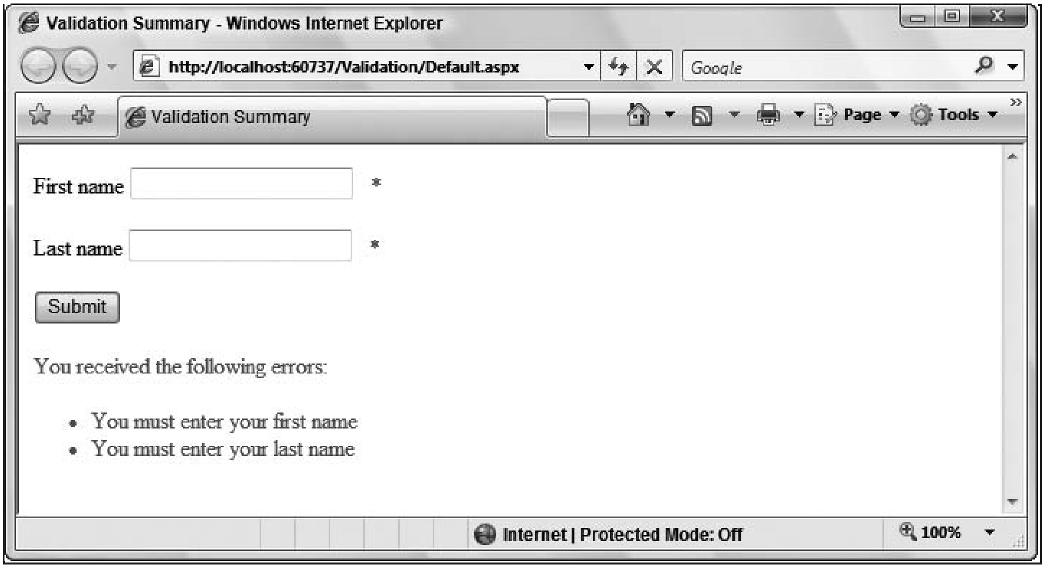
```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 Runat="server" ErrorMessage="You must enter your first name"
 ControlToValidate="TextBox1">*</asp:RequiredFieldValidator>
```

Listing 4-18 shows two ways to accomplish the same task. The first is to use the `Text` property and the second option is to place the provided output between the tags of the `<asp:RequiredFieldValidator>` elements. Making this type of change to the validation controls produces the results shown in Figure 4-9.

To get this result, just remember that the ValidationSummary control uses the validation control's `ErrorMessage` property for displaying the validation errors if they occur. The `Text` property is used by the validation control and is not utilized at all by the ValidationSummary control.

In addition to bulleted lists, you can use the `DisplayMode` property of the ValidationSummary control to change the display of the results to other types of formats. This control has the following possible values:

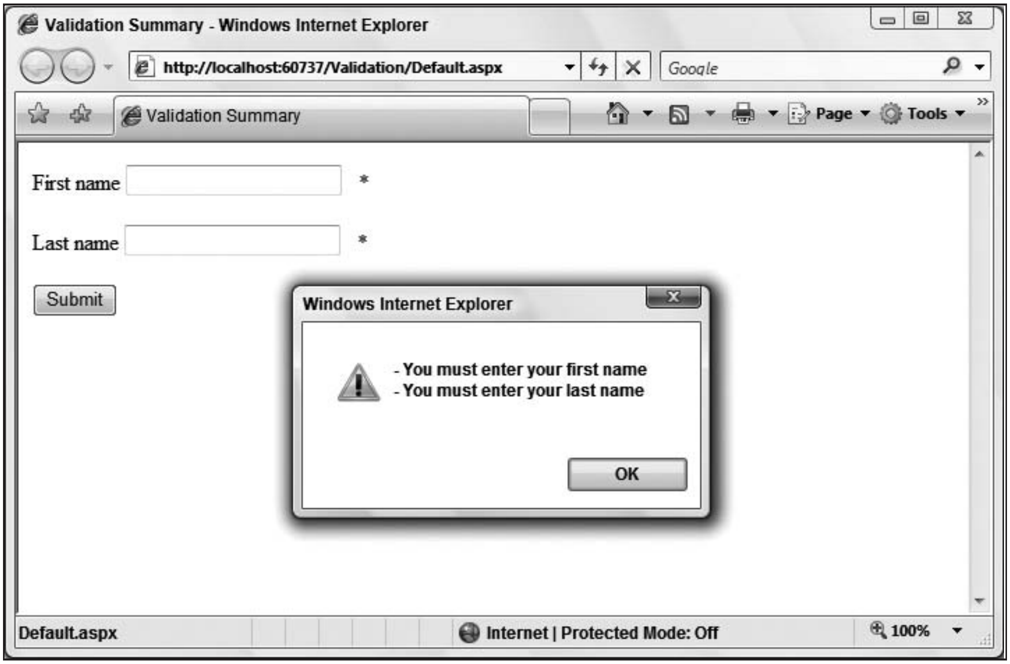
- ❑ `BulletList`
- ❑ `List`
- ❑ `SingleParagraph`


Validation Summary - Windows Internet Explorer
http://localhost:60737/Validation/Default.aspx
Google
Validation Summary
Page
Tools
First name
*
Last name
*
Submit
Internet | Protected Mode: Off
100%

You received the following errors:

- You must enter your first name
- You must enter your last name

Figure 4-9


Validation Summary - Windows Internet Explorer
http://localhost:60737/Validation/Default.aspx
Google
Validation Summary
Page
Tools
First name
*
Last name
*
Submit
Windows Internet Explorer
OK
Default.aspx
Internet | Protected Mode: Off
100%

- You must enter your first name
- You must enter your last name

Figure 4-10

You can also utilize a dialog box instead of displaying the results to the Web page. Listing 4-19 shows an example of this behavior.

Listing 4-19: Using a dialog box to report validation errors

```
<asp:ValidationSummary ID="ValidationSummary1" Runat="server"
 ShowMessageBox="True" ShowSummary="False"></asp:ValidationSummary>
```

From this code example, you can see that the `ShowSummary` property is set to `False` — meaning that the bulleted list of validation errors are not shown on the actual Web page. However, because the `ShowMessageBox` property is set to `True`, you now get these errors reported in a message box, as illustrated in Figure 4-10.

Turning Off Client-Side Validation

Because validation server controls provide clients with client-side validations automatically (if the requesting container can properly handle the JavaScript produced), you might, at times, want a way to control this behavior.

It is quite possible to turn off the client-side capabilities of these controls so that they don't independently send client-side capabilities to the requestors. For instance, you might want all validations done on the server, no matter what capabilities the requesting containers offer. You can take a couple of approaches to turning off this functionality.

The first is at the control level. Each of the validation server controls has a property called `EnableClientScript`. This property is set to `True` by default, but setting it to `False` prevents the control from sending out a JavaScript function for validation on the client. Instead, the validation check is done on the server. The use of this property is shown in Listing 4-20.

Listing 4-20: Disabling client-side validations in a validation control

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" Runat="server"
 Text="*" ControlToValidate="TextBox1" EnableClientScript="false" />
```

You can also remove a validation control's client-side capability programmatically (shown in Listing 4-21).

Listing 4-21: Removing the client-side capabilities programmatically

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   RequiredFieldValidator1.EnableClientScript = False
End Sub
```

C#

```
protected void Page_Load(Object sender, EventArgs e) {
   RequiredFieldValidator1.EnableClientScript = false;
}
```

Another option is to turn off the client-side script capabilities for all the validation controls on a page from within the `Page_Load` event. This can be rather helpful if you want to dynamically decide not to allow client-side validation. This is illustrated in Listing 4-22.

Listing 4-22: Disabling all client-side validations from the Page_Load event

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   For Each bv As BaseValidator In Page.Validators
      bv.EnableClientScript = False
   Next
End Sub
```

C#

```
protected void Page_Load(Object sender, EventArgs e) {
   foreach(BaseValidator bv in Page.Validators)
   {
      bv.EnableClientScript = false;
   }
}
```

Looking for each instance of a `BaseValidator` object in the validators contained on an ASP.NET page, this `For Each` loop turns off client-side validation capabilities for each and every validation control the page contains.

Using Images and Sounds for Error Notifications

So far, we have been displaying simple textual messages for the error notifications that come from the validation server controls. In most instances, you are going to do just that — display some simple textual messages to inform end users that they input something into the form that doesn't pass your validation rules.

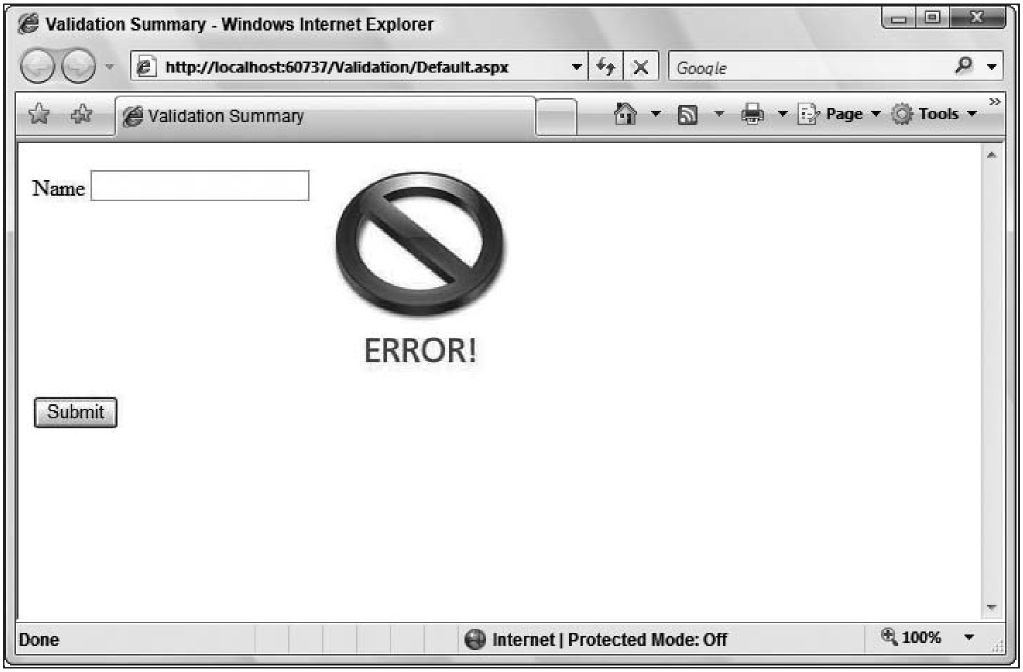
An interesting tip regarding the validation controls is that you are not limited to just text — you can also use images and sounds for error notifications.

To do this, you use the `Text` property of any of the validation controls. To use an image for the error, you can simply place some appropriate HTML as the value of this property. This is illustrated in Listing 4-23.

Listing 4-23: Using images for error notifications

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 Runat="server" Text='<img src="error.gif">'
 ControlToValidate="TextBox1"></asp:RequiredFieldValidator>
```

As you can see from this example, instead of some text being output to the Web page, the value of the `Text` property is an HTML string. This bit of HTML is used to display an image. Be sure to notice the use of the single and double quotation marks so you won't get any errors when the page is generated in the browser. This example produces something similar to what is shown in Figure 4-11.



Validation Summary - Windows Internet Explorer
http://localhost:60737/Validation/Default.aspx
Google
Validation Summary
Page
Tools
Name
ERROR!
Submit
Done
Internet | Protected Mode: Off
100%

Figure 4-11

The other interesting twist you can create is to add a sound notification when the end user errs. You can do this the same way you display an image for error notifications. Listing 4-24 shows an example of this.

Listing 4-24: Using sound for error notifications

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 Runat="server" Text='<bgsound src="C:\Windows\Media\tada.wav">'
 ControlToValidate="TextBox1" EnableClientScript="False">
</asp:RequiredFieldValidator>
```

You can find a lot of the Windows system sounds in the `C:\Windows\Media` directory. In this example, the `Text` uses the `<bgsound>` element to place a sound on the Web form (works only with Internet Explorer). The sound is played only when the end user triggers the validation control.

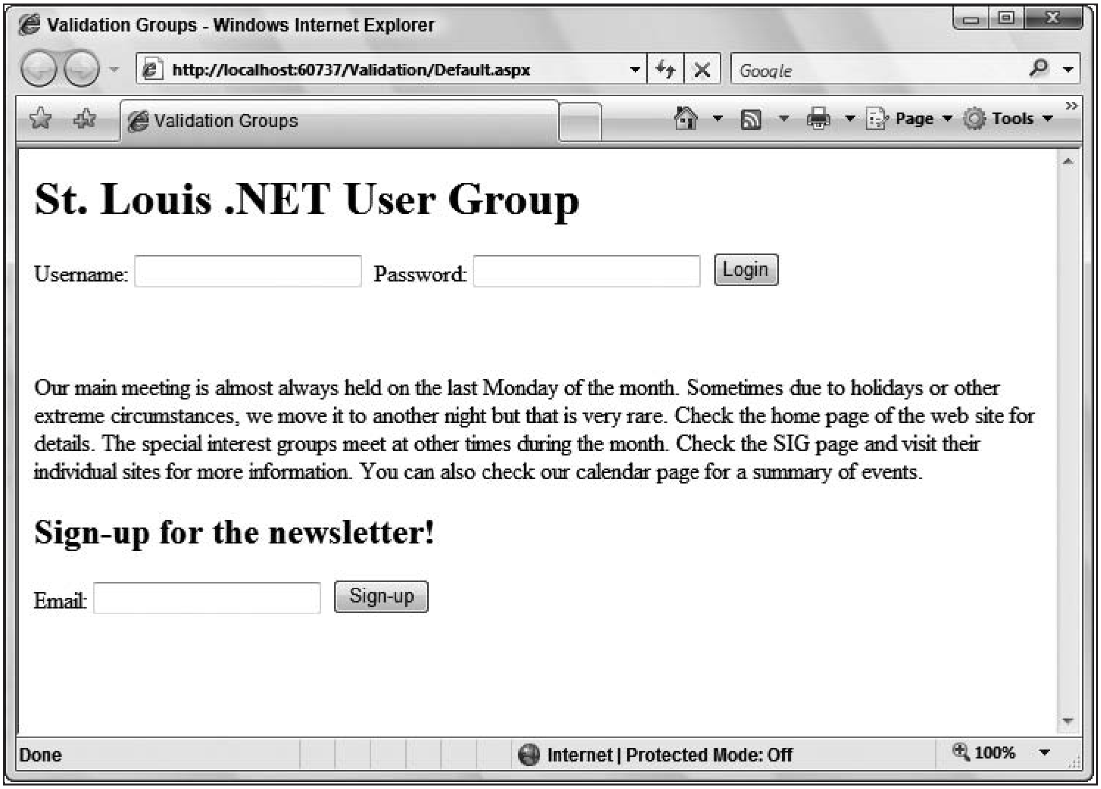
When working with sounds for error notifications, you have to disable the client-side script capability for that particular control because if you do not, the sound plays when the page is loaded in the browser, whether or not a validation error has been triggered.

Working with Validation Groups

In many instances, developers want to place more than one form on a single page. This was always possible in ASP.NET 1.0/1.1 because different button clicks could be used to perform different server-side events. Some issues related to this type of construction were problematic, however.

One of these issues was the difficulty of having validation controls for each of the forms on the page. Different validation controls were often assigned to two distinct forms on the page. When the end user submitted one form, the validation controls in the other form were fired (because the user was not working with that form), thereby stopping the first form from being submitted.

Figure 4-12, for example, shows a basic page for the St. Louis .NET User Group that includes two forms.


Validation Groups - Windows Internet Explorer
http://localhost:60737/Validation/Default.aspx
Google
Validation Groups
Page
Tools
Done
Internet | Protected Mode: Off
100%

St. Louis .NET User Group

Username: Password: Login

Our main meeting is almost always held on the last Monday of the month. Sometimes due to holidays or other extreme circumstances, we move it to another night but that is very rare. Check the home page of the web site for details. The special interest groups meet at other times during the month. Check the SIG page and visit their individual sites for more information. You can also check our calendar page for a summary of events.

Sign-up for the newsletter!

Email: Sign-up

Figure 4-12

One of the forms is for members of the site to supply their usernames and passwords to log into the Members Only section of the site. The second form on the page is for anyone who wishes to sign up for the user group's newsletter. Each form has its own button and some validation controls associated with it. The problem arises when someone submits information for one of the forms. For instance, if you were a member of the group, you would supply your username and password, and click the Login button. The validation controls for the newsletter form would fire because no e-mail address was placed in that particular form. If someone interested in getting the newsletter places an e-mail address in the last text box and clicks the Sign-up button, the validation controls in the first form fire because no username and password were input in that form.

ASP.NET 3.5 provides you with a `ValidationGroup` property that enables you to separate the validation controls into separate groups. It enables you to activate only the required validation controls when an end user clicks a button on the page. Listing 4-25 shows an example of separating the validation controls on a user group page into different buckets.

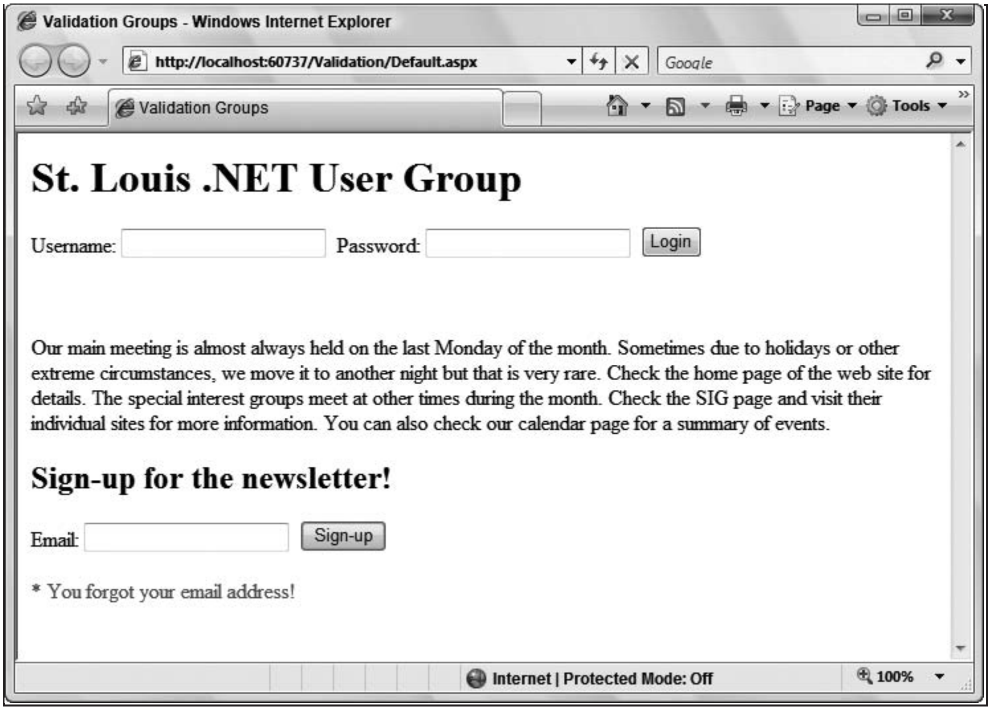
Listing 4-25: Using the ValidationGroup property

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Validation Groups</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <h1>St. Louis .NET User Group</h1>
        <p>Username:
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>&nbsp; Password:
        <asp:TextBox ID="TextBox2" Runat="server"
         TextMode="Password"></asp:TextBox>&nbsp;
        <asp:Button ID="Button1" Runat="server" Text="Login"
         ValidationGroup="Login" />
            <br />
            <asp:RequiredFieldValidator ID="RequiredFieldValidator1" Runat="server"
             Text="* You must submit a username!"
             ControlToValidate="TextBox1" ValidationGroup="Login">
            </asp:RequiredFieldValidator>
            <br />
            <asp:RequiredFieldValidator ID="RequiredFieldValidator2" Runat="server"
             Text="* You must submit a password!"
             ControlToValidate="TextBox2" ValidationGroup="Login">
            </asp:RequiredFieldValidator>
        <p>
            Our main meeting is almost always held on the last Monday of the month.
            Sometimes due to holidays or other extreme circumstances,
            we move it to another night but that is very rare. Check the home page
            of the web site for details. The special
            interest groups meet at other times during the month. Check the SIG
            page and visit their individual sites for more information.
            You can also check our calendar page for a summary of events.<br />
        </p>
        <h2>Sign-up for the newsletter!</h2>
        <p>Email:
        <asp:TextBox ID="TextBox3" Runat="server"></asp:TextBox>&nbsp;
        <asp:Button ID="Button2" Runat="server" Text="Sign-up"
         ValidationGroup="Newsletter" />&nbsp;
            <br />
            <asp:RegularExpressionValidator ID="RegularExpressionValidator1"
             Runat="server"
             Text="* You must submit a correctly formatted email address!"
             ControlToValidate="TextBox3" ValidationGroup="Newsletter"
             ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
            </asp:RegularExpressionValidator>
            <br />
            <asp:RequiredFieldValidator ID="RequiredFieldValidator3" Runat="server"
             Text="* You forgot your email address!"
             ControlToValidate="TextBox3" ValidationGroup="Newsletter">
            </asp:RequiredFieldValidator>
        </p>
    </div>
    </form>
</body>
</html>
```

The `ValidationGroup` property on this page is shown in bold. You can see that this property takes a `String` value. Also note that not only validation controls have this new property. The core server controls also have the `ValidationGroup` property because things like button clicks must be associated with specific validation groups.

In this example, each of the buttons has a distinct validation group assignment. The first button on the form uses `Login` as a value, and the second button on the form uses `Newsletter` as a value. Then each of the validation controls is associated with one of these validation groups. Because of this, when the end user clicks the Login button on the page, ASP.NET recognizes that it should work only with the validation server controls that have the same validation group name. ASP.NET ignores the validation controls assigned to other validation groups.

Using this enhancement, you can now have multiple sets of validation rules that fire only when you want them to fire (see Figure 4-13).


Validation Groups - Windows Internet Explorer
http://localhost:60737/Validation/Default.aspx
Google
Validation Groups
Page
Tools
Login
Sign-up
Internet | Protected Mode: Off
100%

St. Louis .NET User Group

Username: Password:

Our main meeting is almost always held on the last Monday of the month. Sometimes due to holidays or other extreme circumstances, we move it to another night but that is very rare. Check the home page of the web site for details. The special interest groups meet at other times during the month. Check the SIG page and visit their individual sites for more information. You can also check our calendar page for a summary of events.

Sign-up for the newsletter!

Email:

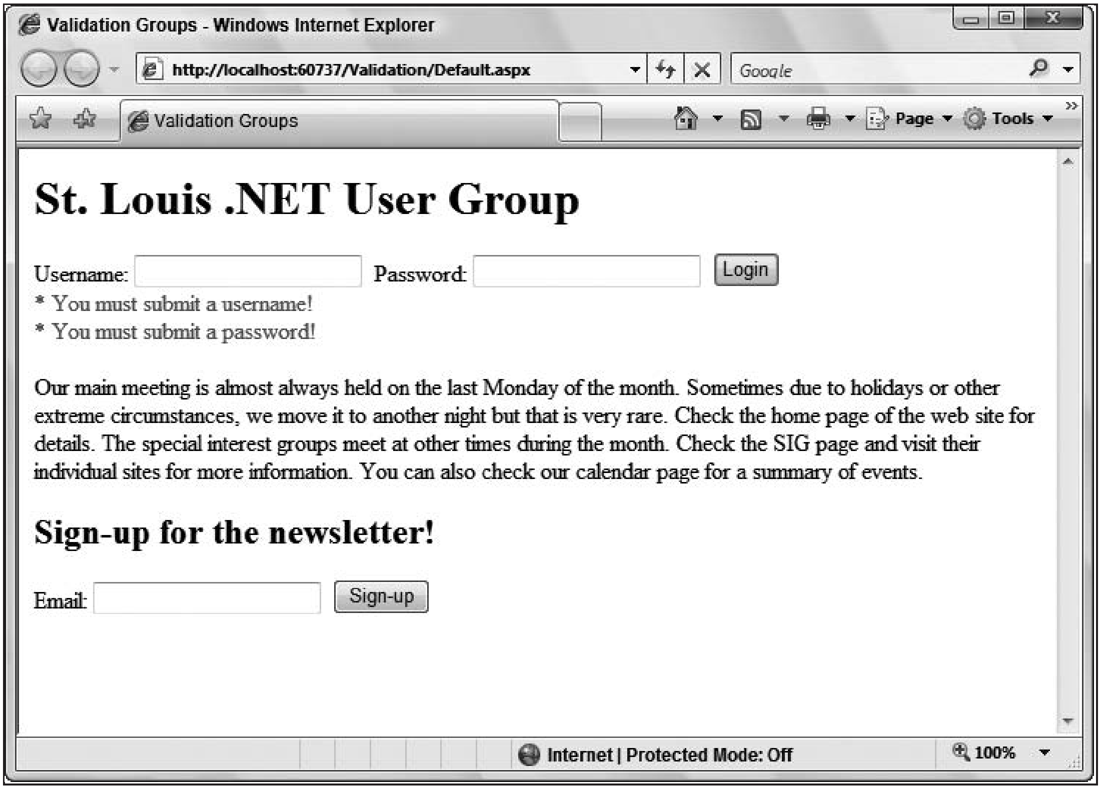
* You forgot your email address!

Figure 4-13

Another great feature that has been added to validation controls is a property called `SetFocusOnError`. This property takes a `Boolean` value and, if a validation error is thrown when the form is submitted, the property places the page focus on the form element that receives the error. The `SetFocusOnError` property is used in the following example:

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" Runat="server"
 Text="* You must submit a username!"
 ControlToValidate="TextBox1" ValidationGroup="Login" SetFocusOnError="True">
</asp:RequiredFieldValidator>
```

If `RequiredFieldValidator1` throws an error because the end user didn't place a value in `TextBox1`, the page is redrawn with the focus on `TextBox1`, as shown in Figure 4-14.


Validation Groups - Windows Internet Explorer
http://localhost:60737/Validation/Default.aspx
Google
Validation Groups
Page
Tools
Internet | Protected Mode: Off
100%

St. Louis .NET User Group

Username: Password: Login

* You must submit a username!

* You must submit a password!

Our main meeting is almost always held on the last Monday of the month. Sometimes due to holidays or other extreme circumstances, we move it to another night but that is very rare. Check the home page of the web site for details. The special interest groups meet at other times during the month. Check the SIG page and visit their individual sites for more information. You can also check our calendar page for a summary of events.

Sign-up for the newsletter!

Email: Sign-up

Figure 4-14

Note that if you have multiple validation controls on your page with the `SetFocusOnError` property set to `True`, and more than one validation error occurs, the uppermost form element that has a validation error gets the focus. In the previous example, if both the username text box (`TextBox1`) and the password text box (`TextBox2`) have validation errors associated with them, the page focus is assigned to the username text box because it is the first control on the form with an error.

Summary

Validation controls are a welcome addition for those developers moving from Active Server Pages to ASP.NET. They bring a lot of functionality in a simple-to-use package and, like most things in the .NET world, you can easily get them to look and behave exactly as you want them to.

Remember that the purpose of having forms in your applications is to collect data, but this data collection has no meaning if the data is not valid. This means that you must establish validation rules that can be implemented in your forms through a series of different controls — the validation server controls.

This chapter covered various validation controls in detail, including

- ☐ RequiredFieldValidator
- ☐ CompareValidator
- ☐ RangeValidator
- ☐ RegularExpressionValidator
- ☐ CustomValidator
- ☐ ValidationSummary

In addition to looking at the base validation controls, this chapter also discussed how to apply client-side and server-side validations.

5

Working with Master Pages

Visual inheritance is a great new enhancement to your Web pages provided by ASP.NET 3.5. This feature was introduced to ASP.NET in version 2.0. In effect, you can create a single template page that can be used as a foundation for any number of ASP.NET content pages in your application. These templates, called *master pages*, increase your productivity by making your applications easier to build and easier to manage after they are built. Visual Studio 2008 includes full designer support for master pages, making the developer experience richer than ever before. This chapter takes a close look at how to utilize master pages to the fullest extent in your applications and begins by explaining the advantages of master pages.

Why Do You Need Master Pages?

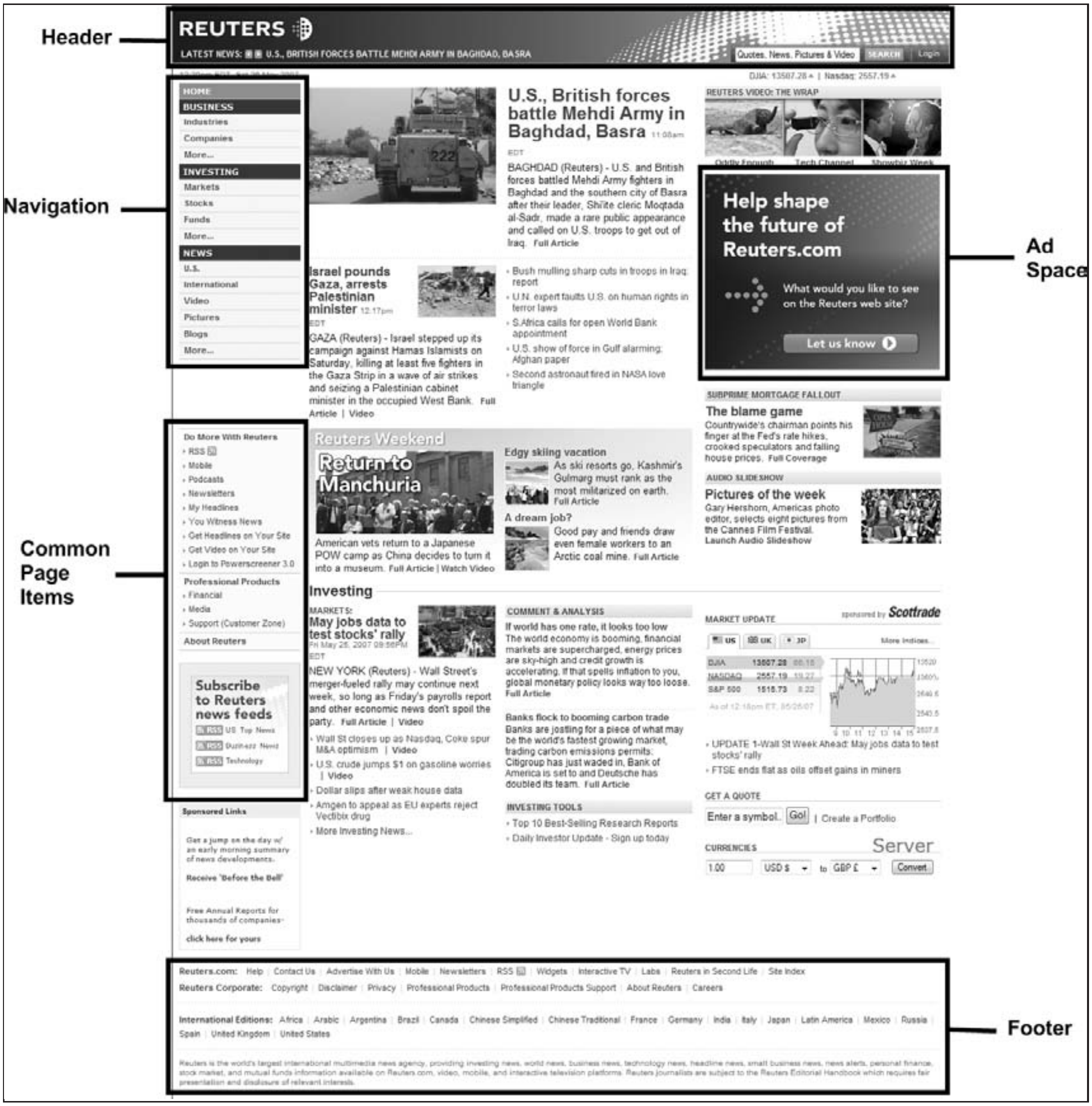
Most Web sites today have common elements used throughout the entire application or on a majority of the pages within the application. For instance, if you look at the main page of the Reuters News Web site (found at `www.reuters.com`), you see common elements that are used throughout the entire Web site. These common areas are labeled in Figure 5-1.

In this screen shot, notice the header section, the navigation section, and the footer section on the page. In fact, nearly every page within the entire application uses these same elements. Even before master pages, you had ways to put these elements into every page through a variety of means; but in most cases, doing so posed difficulties.

Some developers simply copy and paste the code for these common sections to each and every page that requires them. This works, but it's rather labor intensive. However, if you use the copy-and-paste method, whenever a change is required to one of these common sections of the application, you have to go into each and every page and duplicate the change. That's not much fun and an ineffective use of your time!

In the days of Classic Active Server Pages, one popular option was to put all the common sections into what was called an *include file*. You could then place this file within your page like this:

```
<!-- #include virtual="/myIncludes/header.asp" -->
```



REUTERS

LATEST NEWS: U.S., BRITISH FORCES BATTLE MEHDI ARMY IN BAGHDAD, BASRA

Quotes, News, Pictures & Video SEARCH | Login

DJIA: 13507.28 ▲ | Nasdaq: 2557.19 ▲

HOME
BUSINESS
Industries
Companies
More...
INVESTING
Markets
Stocks
Funds
More...
NEWS
U.S.
International
Video
Pictures
Blogs
More...

U.S., British forces battle Mehdi Army in Baghdad, Basra 11:08am EDT

BAGHDAD (Reuters) - U.S. and British forces battled Mehdi Army fighters in Baghdad and the southern city of Basra after their leader, Shi'ite cleric Moqtada al-Sadr, made a rare public appearance and called on U.S. troops to get out of Iraq. **Full Article**

Israel pounds Gaza, arrests Palestinian minister 12:17pm EDT

GAZA (Reuters) - Israel stepped up its campaign against Hamas Islamists on Saturday, killing at least five fighters in the Gaza Strip in a wave of air strikes and seizing a Palestinian cabinet minister in the occupied West Bank. **Full Article | Video**

- Bush mulling sharp cuts in troops in Iraq: report
- U.N. expert faults U.S. on human rights in terror laws
- S.Africa calls for open World Bank appointment
- U.S. show of force in Gulf alarming: Afghan paper
- Second astronaut fired in NASA love triangle

REUTERS VIDEO: THE WRAP

Oddly Enough | Tech Channel | Showbiz Week

Help shape the future of Reuters.com

What would you like to see on the Reuters web site?

Let us know

SUBPRIME MORTGAGE FALLOUT

The blame game

Countrywide's chairman points his finger at the Fed's rate hikes, crooked speculators and falling house prices. **Full Coverage**

AUDIO SLIDESHOW

Pictures of the week

Gary Hershorn, Americas photo editor, selects eight pictures from the Cannes Film Festival.
Launch Audio Slideshow

Do More With Reuters
- RSS
- Mobile
- Podcasts
- Newsletters
- My Headlines
- You Witness News
- Get Headlines on Your Site
- Get Video on Your Site
- Login to Powerscreener 3.0

Professional Products
- Financial
- Media
- Support (Customer Zone)

About Reuters

Subscribe to Reuters news feeds
RSS US Top News
RSS Business News
RSS Technology

Sponsored Links

Get a jump on the day w/ an early morning summary of news developments.
Receive 'Before the Bell'

Free Annual Reports for thousands of companies-
click here for yours

Reuters Weekend

Return to Manchuria

American vets return to a Japanese POW camp as China decides to turn it into a museum. **Full Article | Watch Video**

Edgy skiing vacation

As ski resorts go, Kashmir's Gulmarg must rank as the most militarized on earth. **Full Article**

A dream job?

Good pay and friends draw even female workers to an Arctic coal mine. **Full Article**

Investing

MARKETS:

May jobs data to test stocks' rally Fri May 25, 2007 09:56PM EDT

NEW YORK (Reuters) - Wall Street's merger-fueled rally may continue next week, so long as Friday's payrolls report and other economic news don't spoil the party. **Full Article | Video**

- Wall St closes up as Nasdaq, Coke spur M&A optimism | Video
- U.S. crude jumps $1 on gasoline worries | Video
- Dollar slips after weak house data
- Amgen to appeal as EU experts reject Vectibix drug
- More Investing News...

COMMENT & ANALYSIS

If world has one rate, it looks too low

The world economy is booming, financial markets are supercharged, energy prices are sky-high and credit growth is accelerating. If that spells inflation to you, global monetary policy looks way too loose. **Full Article**

Banks flock to booming carbon trade

Banks are jostling for a piece of what may be the world's fastest growing market, trading carbon emissions permits: Citigroup has just waded in, Bank of America is set to and Deutsche has doubled its team. **Full Article**

INVESTING TOOLS
- Top 10 Best-Selling Research Reports
- Daily Investor Update - Sign up today

MARKET UPDATE sponsored by Scottrade

US | UK | JP More Indices...

DJIA	13507.28	66.15
NASDAQ	2557.19	19.27
S&P 500	1515.73	8.22

As of 12:18pm ET, 05/26/07

- UPDATE 1-Wall St Week Ahead: May jobs data to test stocks' rally
- FTSE ends flat as oils offset gains in miners

GET A QUOTE

Enter a symbol. Go! | Create a Portfolio

CURRENCIES Server

1.00 USD $ to GBP £ Convert

Reuters.com: Help | Contact Us | Advertise With Us | Mobile | Newsletters | RSS | Widgets | Interactive TV | Labs | Reuters in Second Life | Site Index

Reuters Corporate: Copyright | Disclaimer | Privacy | Professional Products | Professional Products Support | About Reuters | Careers

International Editions: Africa | Arabic | Argentina | Brazil | Canada | Chinese Simplified | Chinese Traditional | France | Germany | India | Italy | Japan | Latin America | Mexico | Russia | Spain | United Kingdom | United States

Reuters is the world's largest international multimedia news agency, providing investing news, world news, business news, technology news, headline news, small business news, news alerts, personal finance, stock market, and mutual funds information available on Reuters.com, video, mobile, and interactive television platforms. Reuters journalists are subject to the Reuters Editorial Handbook which requires fair presentation and disclosure of relevant interests.

Figure 5-1

The problem with using `include` files was that you had to take into account the newly opened HTML tags in the header `include` file. These tags had to be closed in the main document or in the footer `include` file. It was usually difficult to keep all the HTML tags in order, especially if multiple people worked on a project. Web pages sometimes displayed strange results because of inappropriate or nonexistent tag closings or openings. It was also difficult to work with `include` files in a visual designer. Using `include` files didn't allow the developer to see the entire page as it would appear in a browser. The developer ended up developing the page in sections and *hoping* that the pieces would come together as planned. Many hours were wasted ''chasing tables'' opened in an include file and possibly closed later!

With the introduction of ASP.NET 1.0 in 2000, developers started using *user controls* to encapsulate common sections of their Web pages. For instance, you could build a Web page that included header, navigation, and footer sections by simply dragging and dropping these sections of code onto each page that required them.

This technique worked, but it also raised some issues. Before Visual Studio 2005 and ASP.NET 2.0, user controls caused problems similar to those related to `include` files. When you worked in the design view of your Web page, the common areas of the page displayed only as gray boxes in Visual Studio .NET 2002 and 2003. This made it harder to build a page. You could not visualize what the page you were building actually looked like until you compiled and ran the completed page in a browser. User controls also suffered from the same problem as `include` files — you had to match up the opening and closing of your HTML tags in two separate files. Personally, we prefer user controls over `include` files, but user controls aren't perfect template pieces for use throughout an application. You will find that Visual Studio corrects some of the problems by rendering user-control content in the design view. User controls are ideal if you are including only small sections on a Web page; they are still rather cumbersome, however, when working with larger page templates.

In light of the issues with `include` files and user controls, the ASP.NET team developed the idea of *master pages* — an outstanding new way of applying templates to your applications. They *inverted* the way the developer attacks the problem. Master pages live *outside* the pages you develop, while user controls live within your pages and are doomed to duplication. These master pages draw a more distinct line between the common areas that you carry over from page to page and the content areas that are unique on each page. You will find that working with master pages is easy and fun. The next section of this chapter looks at some of the basics of master pages in ASP.NET.

The Basics of Master Pages

Master pages are an easy way to provide a template that can be used by any number of ASP.NET pages in your application. In working with master pages, you create a master file that is the template referenced by a *subpage* or *content page*. Master pages use a `.master` file extension, whereas content pages use the `.aspx` file extension you're used to; but content pages are declared as such within the file's `Page` directive.

You can place anything you want to be included as part of the template in the `.master` file. This can include the header, navigation, and footer sections used across the Web application. The content page then contains all the page content except for the master page's elements. At runtime, the ASP.NET engine combines these elements into a single page for the end user. Figure 5-2 shows a diagram of how this process works.

One of the nice things about working with master pages is that you can visually see the template in the IDE when you are creating the content pages. Because you can see the entire page while you are working on it, it is much easier to develop content pages that use a template. While you are working on the content page, all the templated items are shaded gray and are not editable. The only items that are alterable are clearly shown in the template. These workable areas, called *content areas*, originally are defined in the master page itself. Within the master page, you specify the areas of the page that the content pages can use. You can have more than one content area in your master page if you want. Figure 5-3 shows the master page with a couple of content areas shown.

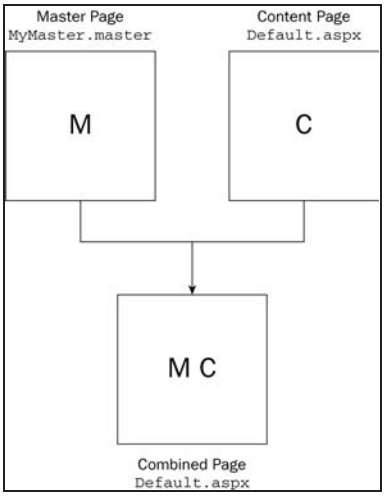
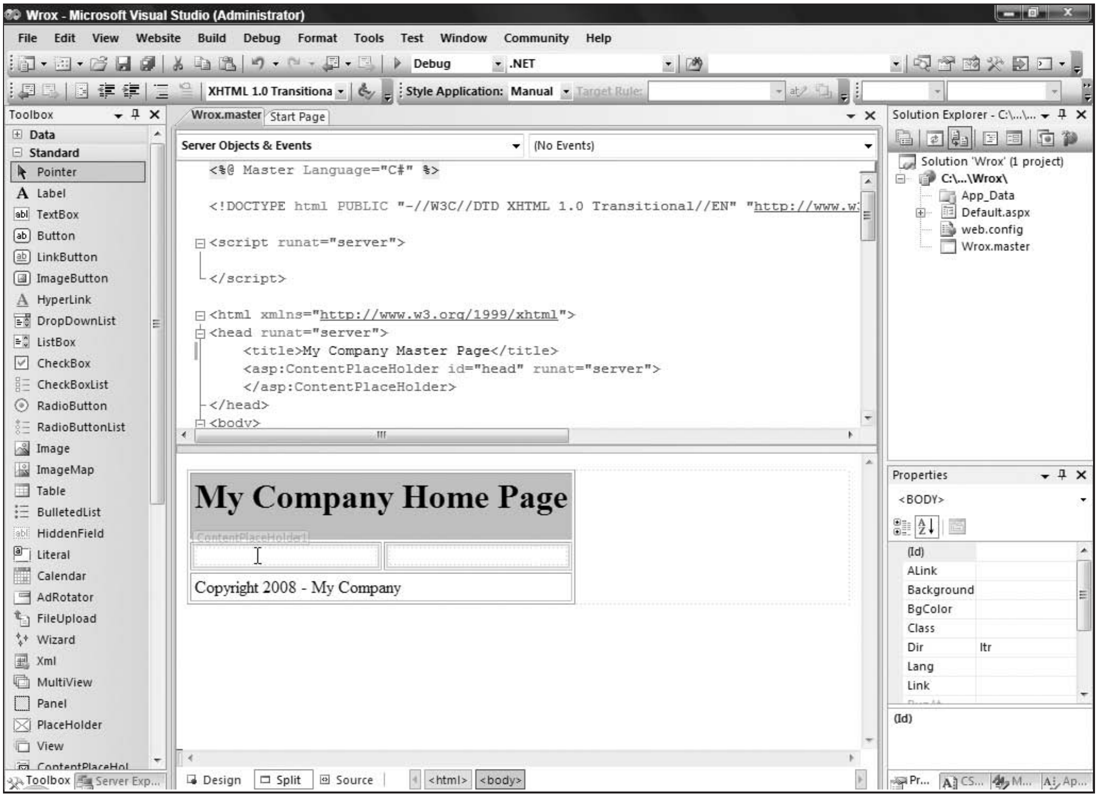

Master Page
MyMaster.master
Content Page
Default.aspx
M
C
M C
Combined Page
Default.aspx

Figure 5-2


Wrox - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Format Tools Test Window Community Help
Debug
.NET
XHTML 1.0 Transitiona
Style Application: Manual
Target Rule:
Toolbox
Data
Standard
Pointer
Label
TextBox
Button
LinkButton
ImageButton
HyperLink
DropDownList
ListBox
CheckBox
CheckBoxList
RadioButton
RadioButtonList
Image
ImageMap
Table
BulletedList
HiddenField
Literal
Calendar
AdRotator
FileUpload
Wizard
Xml
MultiView
Panel
PlaceHolder
View
ContentPlaceHol
Toolbox
Server Exp...
Wrox.master
Start Page
Server Objects & Events
(No Events)
Solution Explorer - C:\...\...
Solution 'Wrox' (1 project)
C:\...\Wrox\
App_Data
Default.aspx
web.config
Wrox.master
Properties
<BODY>
(Id)
ALink
Background
BgColor
Class
Dir
ltr
Lang
Link
(Id)
Design
Split
Source
<html>
<body>
Pr...
CS...
M...
Ap...

```
<%@ Master Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>My Company Master Page</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
```

My Company Home Page

ContentPlaceHolder1

Copyright 2008 - My Company

Figure 5-3

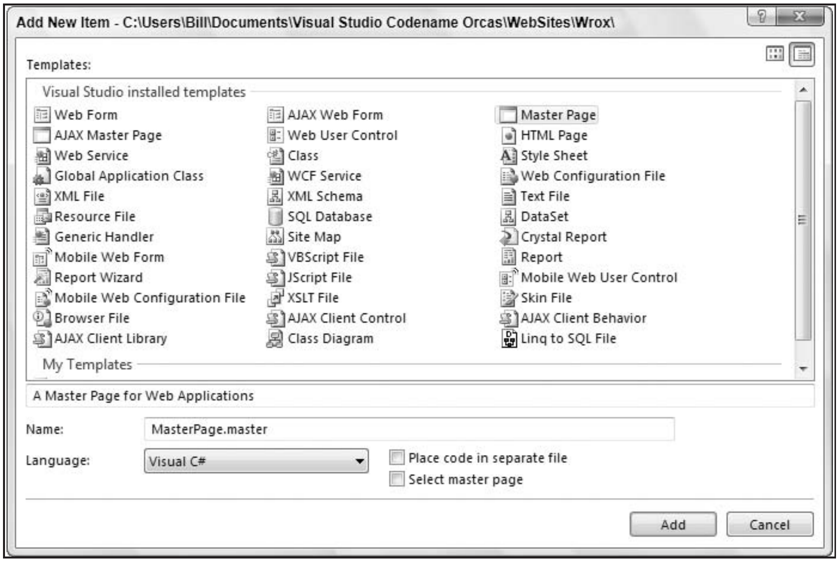
If you look at the screenshot from Figure 5-3, you can sort of see two defined areas on the page — these are content areas. The content area is represented in the design view of the page by a light dotted box that represents the ContentPlaceHolder control. Also, if you hover your mouse over the content area, you will see the name of the control appear above the control (although lightly). This hovering is also shown in action in Figure 5-3.

Companies and organizations will find using master pages ideal, as the technology closely models their typical business requirements. Many companies have a common look and feel that they apply across their intranet. They can now provide the divisions of their company with a `.master` file to use when creating a department's section of the intranet. This process makes it quite easy for the company to keep a consistent look and feel across its entire intranet.

Coding a Master Page

Now look at building the master page shown previously in Figure 5-3. You can create one in any text-based editor, such as Notepad or Visual Web Developer Express Edition, or you can use the new Visual Studio 2008. In this chapter, you see how it is done with Visual Studio 2008.

Master pages are added to your projects in the same way as regular `.aspx` pages — choose the Master Page option when you add a new file to your application, as shown in Figure 5-4.



Add New Item - C:\Users\Bill\Documents\Visual Studio Codename Orcas\WebSites\Wrox\

A Master Page for Web Applications

Name: MasterPage.master

Language: Visual C#

☐ Place code in separate file

☐ Select master page

Templates:
Visual Studio installed templates
Web Form
AJAX Master Page
Web Service
Global Application Class
XML File
Resource File
Generic Handler
Mobile Web Form
Report Wizard
Mobile Web Configuration File
Browser File
AJAX Client Library
AJAX Web Form
Web User Control
Class
WCF Service
XML Schema
SQL Database
Site Map
VBScript File
JScript File
XSLT File
AJAX Client Control
Class Diagram
Master Page
HTML Page
Style Sheet
Web Configuration File
Text File
DataSet
Crystal Report
Report
Mobile Web User Control
Skin File
AJAX Client Behavior
Linq to SQL File
My Templates
Add
Cancel

Figure 5-4

Because it's quite similar to any other `.aspx` page, the Add New Item dialog enables you to choose from a master page using the inline coding model or a master page that places its code in a separate file. Not placing your server code in a separate file means that you use the inline code model for the page you are creating. This option creates a single `.master` page. Choosing the option to place your code in a separate file means that you use the new code-behind model with the page you are creating. Selecting the check box ''Place code in separate file'' creates a single `.master` page, along with an associated `.master.vb` or

`.master.cs` file. You also have the option of nesting your master page within another master page by selecting the Select master page option, but this is covered later in this chapter.

A sample master page that uses the inline-coding model is shown in Listing 5-1.

Listing 5-1: A sample master page

```
<%@ Master Language="VB" %>

<script runat="server">

</script>


<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>My Company Master Page</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
        <table cellpadding="3" border="1">
            <tr bgcolor="silver">
                <td colspan="2">
                    <h1>My Company Home Page</h1>
                </td>
            </tr>
            <tr>
                <td>
                    <asp:ContentPlaceHolder ID="ContentPlaceHolder1"
                     runat="server">
                    </asp:ContentPlaceHolder>
                </td>
                <td>
                    <asp:ContentPlaceHolder ID="ContentPlaceHolder2"
                     runat="server">
                    </asp:ContentPlaceHolder>
                </td>
            </tr>
            <tr>
                <td colspan="2">
                    Copyright 2008 - My Company
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

This is a simple master page. The great thing about creating master pages in Visual Studio 2008 is that you can work with the master page in code view, but you can also switch over to design view to create your master pages just as any other ASP.NET page.

Start by reviewing the code for the master page. The first line is the directive:

```
<%@ Master Language="VB" %>
```

Instead of using the `Page` directive, as you would with a typical `.aspx` page, you use the `Master` directive for a master page. This master page uses only a single attribute, `Language`. The `Language` attribute's value here is `VB`, but of course, you can also use `C#` if you are building a C# master page.

You code the rest of the master page just as you would any other `.aspx` page. You can use server controls, raw HTML and text, images, events, or anything else you normally would use for any `.aspx` page. This means that your master page can have a `Page_Load` event as well or any other event that you deem appropriate.

In the code shown in Listing 5-1, notice the use of a new server control — the `<asp:ContentPlaceHolder>` control. This control defines the areas of the template where the content page can place its content:

```
<tr>
   <td>
      <asp:ContentPlaceHolder ID="ContentPlaceHolder1"
       runat="server">
      </asp:ContentPlaceHolder>
   </td>
   <td>
      <asp:ContentPlaceHolder ID="ContentPlaceHolder2"
       runat="server">
      </asp:ContentPlaceHolder>
   </td>
</tr>
```

In the case of this master page, two defined areas exist where the content page can place content. Our master page contains a header and a footer area. It also defines two areas in the page where any inheriting content page can place its own content. Look at how a content page uses this master page.

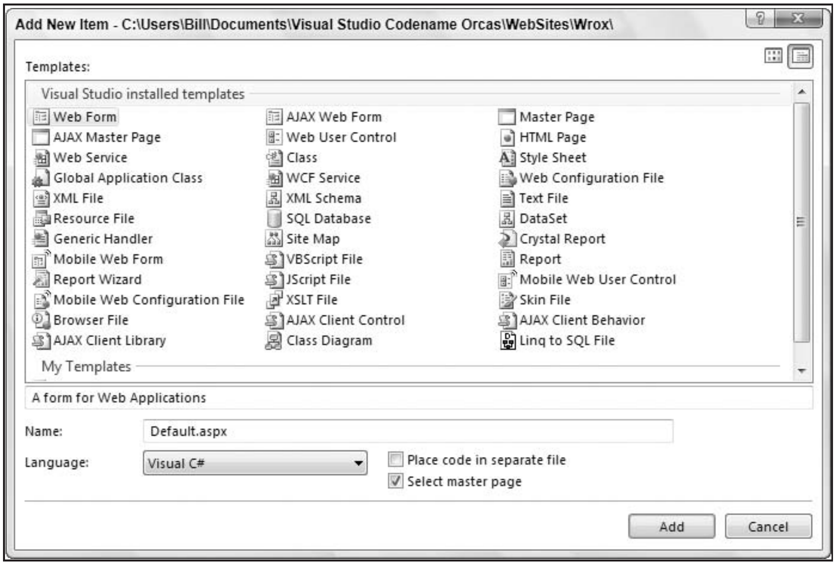
Coding a Content Page

Now that you have a master page in place in your application, you can use this new template for any content pages in your application. Right-click the application in the Solution Explorer and choose Add New Item to create a new content page within your application.

To create a content page or a page that uses this master page as its template, you select a typical Web Form from the list of options in the Add New Item dialog (see Figure 5-5). Instead of creating a typical Web Form, however, you check the Select Master Page check box. This gives you the option of associating this Web Form later to some master page.

After you name your content page and click the Add button in the Add New Item dialog, you are presented with the Select a Master Page dialog, as shown in Figure 5-6.

This dialog allows you to choose the master page from which you want to build your content page. You choose from the available master pages that are contained within your application. For this example, select the new master page that you created in Listing 5-1 and click OK. This creates the content page. The created page is a simple `.aspx` page with only a couple of lines of code contained in the file, as shown in Listing 5-2.


Add New Item - C:\Users\Bill\Documents\Visual Studio Codename Orcas\WebSites\Wrox\
Templates:
Visual Studio installed templates
Web Form
AJAX Master Page
Web Service
Global Application Class
XML File
Resource File
Generic Handler
Mobile Web Form
Report Wizard
Mobile Web Configuration File
Browser File
AJAX Client Library
AJAX Web Form
Web User Control
Class
WCF Service
XML Schema
SQL Database
Site Map
VBScript File
JScript File
XSLT File
AJAX Client Control
Class Diagram
Master Page
HTML Page
Style Sheet
Web Configuration File
Text File
DataSet
Crystal Report
Report
Mobile Web User Control
Skin File
AJAX Client Behavior
Linq to SQL File
My Templates
Add
Cancel

A form for Web Applications

Name: Default.aspx

Language: Visual C#

☐ Place code in separate file

☑ Select master page

Figure 5-5

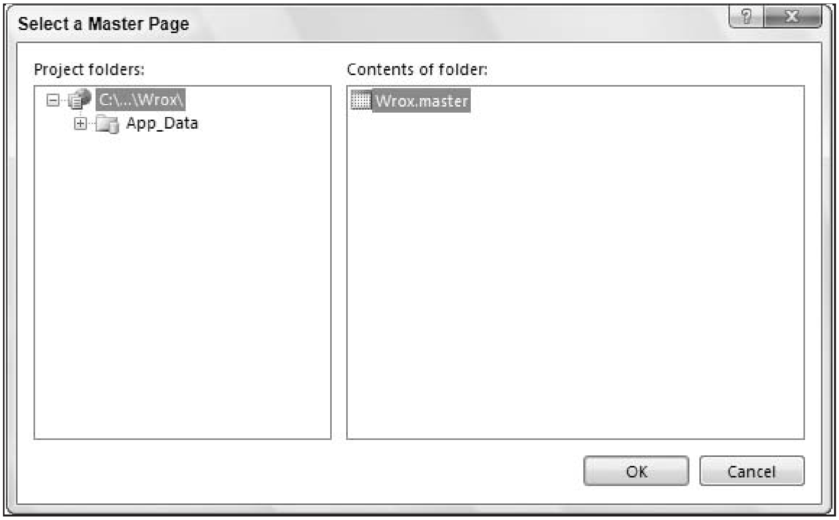

Select a Master Page
Project folders:
C:\...\Wrox\
App_Data
Contents of folder:
Wrox.master
OK
Cancel

Figure 5-6

Listing 5-2: The created content page

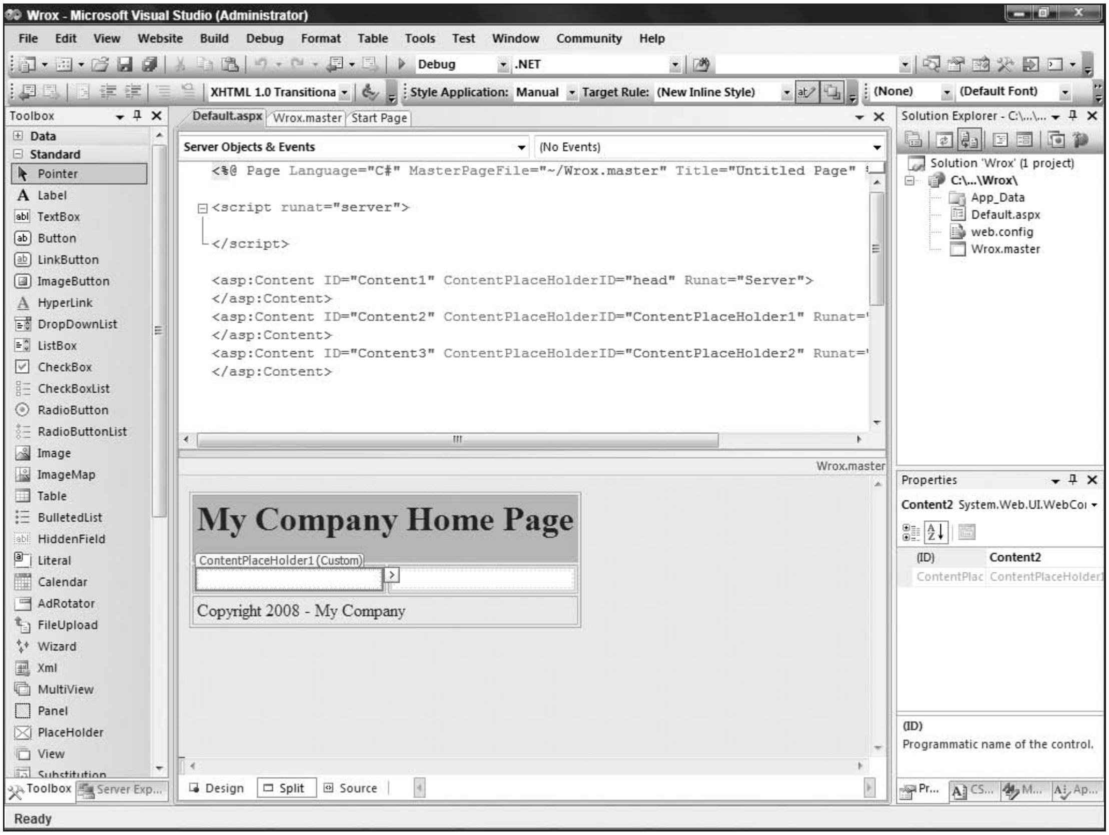
```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" Title="Untitled Page" %>

<script runat="server">

</script>
```

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1"
 Runat="Server">
</asp:Content>
```

This content page is not much different from the typical `.aspx` page you coded in the past. The big difference is the inclusion of the `MasterPageFile` attribute within the `Page` directive. The use of this attribute indicates that this particular `.aspx` page constructs its control's based on another page. The location of the master page within the application is specified as the value of the `MasterPageFile` attribute.

The other big difference is that it contains neither the `<form id="form1" runat="server">` tag nor any opening or closing HTML tags that would normally be included in a typical `.aspx` page.

This content page may seem simple, but if you switch to the design view within Visual Studio 2008, you see the power of using content pages. What you get with visual inheritance is shown in Figure 5-7.


Wrox - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Format Table Tools Test Window Community Help
Debug
.NET
XHTML 1.0 Transitiona
Style Application: Manual
Target Rule: (New Inline Style)
(None)
(Default Font)
Toolbox
Data
Standard
Pointer
Label
TextBox
Button
LinkButton
ImageButton
HyperLink
DropDownList
ListBox
CheckBox
CheckBoxList
RadioButton
RadioButtonList
Image
ImageMap
Table
BulletedList
HiddenField
Literal
Calendar
AdRotator
FileUpload
Wizard
Xml
MultiView
Panel
PlaceHolder
View
Substitution
Toolbox
Server Exp...
Default.aspx
Wrox.master
Start Page
Server Objects & Events
(No Events)
Wrox.master
My Company Home Page
ContentPlaceHolder1 (Custom)
Copyright 2008 - My Company
Design
Split
Source
Solution Explorer - C:\...\...
Solution 'Wrox' (1 project)
C:\...\Wrox\
App_Data
Default.aspx
web.config
Wrox.master
Properties
Content2 System.Web.UI.WebCo
(ID)
Content2
ContentPlac
ContentPlaceHolder
(ID)
Programmatic name of the control.
Pr...
CS...
M...
Ap...
Ready

```
<%@ Page Language="C#" MasterPageFile="~/Wrox.master" Title="Untitled Page"

<script runat="server">

</script>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat='
</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="ContentPlaceHolder2" Runat='
</asp:Content>
```

Figure 5-7

In this screenshot, you can see that just by using the `MasterPageFile` attribute in the `Page` directive, you are able to visually inherit everything that the `Wrox.master` file exposes. From the design view within Visual Studio, you can also see what master page you are working with as the name of the referenced master page is presented in the upper-right corner of the Design view page. If you try and click into the gray area that represents what is inherited from the master page, you will see that your cursor changes to show you are not allowed. This is illustrated in Figure 5-8 (the cursor is on the word Page in the title).

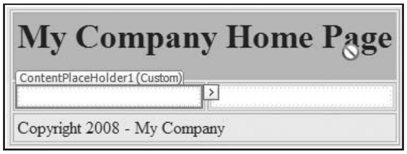


Figure 5-8

All the common areas defined in the master page are shown in gray, whereas the content areas that you specified in the master page using the `<asp:ContentPlaceHolder>` server control are shown clearly and available for additional content in the content page. You can add any content to these defined content areas as if you were working with a regular `.aspx` page. An example of using this `.master` page for a content page is shown in Listing 5-3.

Listing 5-3: The content page that uses Wrox.master

VB

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "Hello " & TextBox1.Text & "!"
   End Sub
</script>

<asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceHolder1"
 runat="server">
    <b>Enter your name:</b><br />
    <asp:Textbox ID="TextBox1" runat="server" />
    <br />
    <br />
    <asp:Button ID="Button1" runat="server" Text="Submit"
     OnClick="Button1_Click" /><br />
    <br />
    <asp:Label ID="Label1" runat="server" Font-Bold="True" />
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderId="ContentPlaceHolder2"
 runat="server">
        <asp:Image ID="Image1" runat="server" ImageUrl="wrox.gif" />
</asp:Content>
```

C#

```
<%@ Page Language="C#" MasterPageFile="~/Wrox.master" %>
```

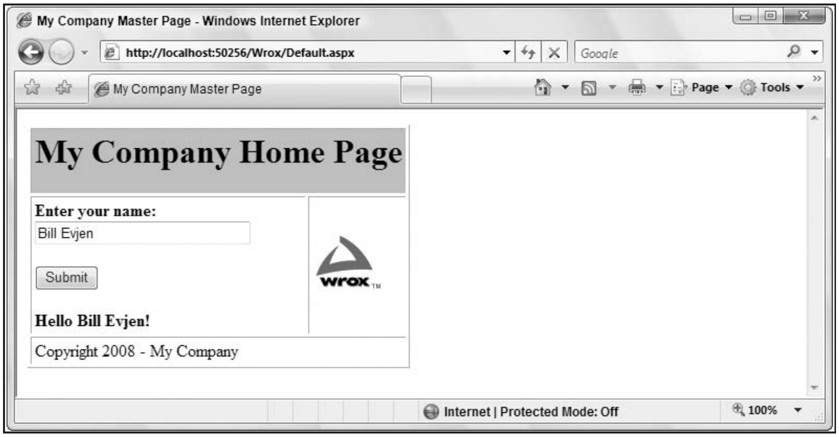
```
<script runat="server">
    protected void Button1_Click(object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + TextBox1.Text + "!";
    }
</script>
```

Right away you see some differences. As stated before, this page has no `<form id="form1" runat="server">` tag nor any opening or closing `<html>` tags. These tags are not included because they are located in the master page. Also notice a new server control — the `<asp:Content>` server control.

```
<asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceHolder1"
 runat="server">
   ...
</asp:Content>
```

The `<asp:Content>` server control is a defined content area that maps to a specific `<asp:ContentPlaceHolder>` server control on the master page. In this example, you can see that the `<asp:Content>` server control maps itself to the `<asp:ContentPlaceHolder>` server control in the master page that has the `ID` of `ContentPlaceHolder1`. Within the content page, you don't have to worry about specifying the location of the content because this is already defined within the master page. Therefore, your only concern is to place the appropriate content within the provided content sections, allowing the master page to do most of the work for you.

Just as when you work with any typical `.aspx` page, you can create any event handlers for your content page. In this case, you are using just a single event handler — the button click when the end user submits the form. The created `.aspx` page that includes the master page and content page material is shown in Figure 5-9.


My Company Master Page - Windows Internet Explorer
http://localhost:50256/Wrox/Default.aspx
Google
My Company Master Page
Page
Tools
Submit
wrox
Internet | Protected Mode: Off
100%

My Company Home Page

Enter your name:

Bill Evjen

Hello Bill Evjen!

Copyright 2008 - My Company

Figure 5-9

Mixing Page Types and Languages

One interesting point: When you use master pages, you are not tying yourself to a specific coding model (inline or code-behind), nor are you tying yourself to the use of a specific language. You can feel free to mix these elements within your application because they all work well.

You could use the master page created earlier, knowing that it was created using the inline-coding model, and then build your content pages using the code-behind model. Listing 5-4 shows a content page created using a Web Form that uses the code-behind option.

Listing 5-4: A content page that uses the code-behind model

.aspx (VB)

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" AutoEventWireup="false"
    CodeFile="MyContentPage.aspx.vb" Inherits="MyContentPage" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderId="ContentPlaceHolder1"
 runat="server">
    <b>Enter your name:</b><br />
    <asp:Textbox ID="TextBox1" runat="server" />
    <br />
    <br />
    <asp:Button ID="Button1" runat="server" Text="Submit" /><br />
    <br />
    <asp:Label ID="Label1" runat="server" Font-Bold="True" />
</asp:Content>

<asp:Content ID="Content3" ContentPlaceHolderId="ContentPlaceHolder2"
 runat="server">
        <asp:Image ID="Image1" runat="server" ImageUrl="ineta.JPG" />
</asp:Content>
```

VB Code-Behind

```
Partial Class MyContentPage
   Inherits System.Web.UI.Page

   Protected Sub Button1_Click(ByVal sender As Object, _
     ByVal e As System.EventArgs) Handles Button1.Click

      Label1.Text = "Hello " & TextBox1.Text & "!"
   End Sub

End Class
```

C# Code-Behind

```
public partial class MyContentPage : System.Web.UI.Page
{
    protected void Button1_Click (object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + TextBox1.Text + "!";
    }
}
```

Even though the master page is using the inline-coding model, you can easily create content pages (such as the page shown in Listing 5-4) that use the code-behind model. The pages will still work perfectly.

Not only can you mix the coding models when using master pages, you can also mix the programming languages you use for the master or content pages. Just because you build a master page in C# doesn't mean that you are required to use C# for all the content pages that use this master page. You can also build content pages in Visual Basic. For a good example, create a master page in C# that uses the `Page_Load` event handler and then create a content page in Visual Basic. Once it is complete, run the page. It works perfectly well. This means that even though you might have a master page in one of the available .NET languages, the programming teams that build applications from the master page can use whatever .NET language they want. You have to love the openness that the .NET Framework˜offers!

Specifying Which Master Page to Use

You just observed that it is pretty easy to specify at page level which master page to use. In the `Page` directive of the content page, you simply use the `MasterPageFile` attribute:

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" %>
```

Besides specifying the master page that you want to use at the page level, you have a second way to specify which master page you want to use in the `web.config` file of the application. This is shown in Listing 5-5.

Listing 5-5: Specifying the master page in the web.config file

```
<configuration>
   <system.web>
      <pages masterPageFile="~/Wrox.master" />
   </system.web>
</configuration>
```

Specifying the master page in the `web.config` file causes every single content page you create in the application to inherit from the specified master page. If you declare your master page in the `web.config` file, you can create any number of content pages that use this master page. Once specified in this manner, the content page's `Page` directive can then be constructed in the following manner:

```
<%@ Page Language="VB" %>
```

You can easily override the application-wide master page specification by simply declaring a different master page within your content page:

```
<%@ Page Language="VB" MasterPageFile="~/MyOtherCompany.master" %>
```

By specifying the master page in the `web.config`, you are really not saying that you want *all* the `.aspx` pages to use this master page. If you create a normal Web Form and run it, ASP.NET will know that the page is not a content page and will run the page as a normal `.aspx` page.

If you want to apply the master page template to only a specific subset of pages (such as pages contained within a specific folder of your application), you can use the `<location>` element within the `web.config` file, as illustrated in Listing 5-6.

Listing 5-6: Specifying the master page for a specific folder in the web.config file

```
<configuration>

    <location path="AdministrationArea">
        <system.web>
           <pages masterPageFile="~/WroxAdmin.master" />
        </system.web>
    </location>

</configuration>
```

With the addition of this `<location>` section in the `web.config` file, you have now specified that a specific folder (`AdministrationArea`) will use a different master file template. This is done using the path attribute of the `<location>` element. The value of the `path` attribute can be a folder name as shown, or it can even be a specific page — such as `AdminPage.aspx`.

Working with the Page Title

When you create content pages in your application, by default all the content pages automatically use the title that is declared in the master page. For instance, you have primarily been using a master page with the title `My Company Master Page`. Every content page that is created using this particular master page also uses the same `My Company Master Page` title. You can avoid this by specifying the page's title using the `Title` attribute in the `@Page` directive in the content page. You can also work with the page title programmatically in your content pages. To accomplish this, in the code of the content page, you use the `Master` object. The `Master` object conveniently has a property called `Title`. The value of this property is the page title that is used for the content page. You code it as shown in Listing 5-7.

Listing 5-7: Coding a custom page title for the content page

VB

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" %>

<script runat="server">
    Protected Sub Page_LoadComplete(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Master.Page.Title = "This page was generated on: " & _
           DateTime.Now.ToString()
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" MasterPageFile="~/wrox.master" %>

<script runat="server">
    protected void Page_LoadComplete(object sender, EventArgs e)
    {
        Master.Page.Title = "This page was generated on: " +
            DateTime.Now.ToString();
    }
</script>
```

Working with Controls and Properties from the Master Page

When working with master pages from a content page, you actually have good access to the controls and the properties that the master page exposes. The master page, when referenced by the content page, exposes a property called `Master`. You use this property to get at control values or custom properties that are contained in the master page itself.

To see an example of this, create a GUID (unique identifier) in the master page that you can retrieve on the content page that is using the master. For this example, use the master page that was created in Listing 5-1, but add a Label server control and the `Page_Load` event (see Listing 5-8).

Listing 5-8: A master page that creates a GUID on the first request

VB

```
<%@ Master Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        If Not Page.IsPostBack Then
            Label1.Text = System.Guid.NewGuid().ToString()
        End If
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>My Company Master Page</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
        <table cellpadding="3" border="1">
            <tr bgcolor="silver">
                <td colspan="2">
                    <h1>My Company Home Page</h1>
                    <b>User's GUID:&nbsp;&nbsp;
                        <asp:Label ID="Label1" runat="server" /></b>
                </td>
            </tr>
            <tr>
                <td>
                    <asp:ContentPlaceHolder ID="ContentPlaceHolder1"
                     runat="server">
                    </asp:ContentPlaceHolder>
                </td>
                <td>
                    <asp:ContentPlaceHolder ID="ContentPlaceHolder2"
                     runat="server">
                    </asp:ContentPlaceHolder>
                </td>
            </tr>
            <tr>
                <td colspan="2">
                    Copyright 2008 - My Company
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

C#

```
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            Label1.Text = System.Guid.NewGuid().ToString();
        }
    }
```

Now you have a Label control on the master page that you can access from the content page. You have a couple of ways to accomplish this task. The first is to use the `FindControl()` method that the master page exposes. This approach is shown in Listing 5-9.

Listing 5-9: Getting at the Label's Text value in the content page

VB

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" %>

<script runat="server" language="vb">
    Protected Sub Page_LoadComplete(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Label1.Text = CType(Master.FindControl("Label1"), Label).Text
    End Sub

    Protected Sub Button1_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Label2.Text = "Hello " & TextBox1.Text & "!"
    End Sub
</script>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderId="ContentPlaceHolder1"
```

```
 runat="server">
    <b>Your GUID number from the master page is:<br />
    <asp:Label ID="Label1" runat="server" /></b><p>
    <b>Enter your name:</b><br />
    <asp:Textbox ID="TextBox1" runat="server" />
    <br />
    <br />
    <asp:Button ID="Button1" runat="server" Text="Submit"
     OnClick="Button1_Click" /><br />
    <br />
    <asp:Label ID="Label2" runat="server" Font-Bold="True" />
</asp:Content>

<asp:Content ID="Content3" ContentPlaceHolderId="ContentPlaceHolder2"
 runat="server">
        <asp:Image ID="Image1" runat="server" ImageUrl="Wrox.gif" />
</asp:Content>
```

C#

```
<%@ Page Language="C#" MasterPageFile="~/wrox.master" %>

<script runat="server">

    protected void Page_LoadComplete(object sender, EventArgs e)
    {
        Label1.Text = (Master.FindControl("Label1") as Label).Text;
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        Label2.Text = "<b>Hello " + TextBox1.Text + "!</b>";
    }
</script>
```

In this example, the master page in Listing 5-8 first creates a GUID that it stores as a text value in a Label server control on the master page itself. The `ID` of this Label control is `Label1`. The master page generates this GUID only on the first request for this particular content page. From here, you then populate one of the content page's controls with this value.

The interesting thing about the content page is that you put code in the `Page_LoadComplete` event handler so that you can get at the GUID value that is on the master page. This new event in ASP.NET fires immediately after the `Page_Load` event fires. Event ordering is covered later, but the `Page_Load` event in the content page always fires before the `Page_Load` event in the master page. In order to get at the newly created GUID (if it is created in the master page's `Page_Load` event), you have to get the GUID in an event that comes after the `Page_Load` event — and that is where the `Page_LoadComplete` comes into play. Therefore, within the content page's `Page_LoadComplete` event, you populate a Label server control within the content page itself. Note that the Label control in the content page has the same `ID` as the Label control in the master page, but this doesn't make a difference. You can differentiate between them with the use of the `Master` property.

Not only can you get at the server controls that are in the master page in this way, you can get at any custom properties that the master page might expose as well. Look at the master page shown in Listing 5-10; it uses a custom property for the `<h1>` section of the page.

Listing 5-10: A master page that exposes a custom property

VB

```
<%@ Master Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        If Not Page.IsPostBack Then
            Label1.Text = Guid.NewGuid().ToString()
        End If
    End Sub

    Dim m_PageHeadingTitle As String = "My Company"

    Public Property PageHeadingTitle() As String
        Get
            Return m_PageHeadingTitle
        End Get
        Set(ByVal Value As String)
            m_PageHeadingTitle = Value
        End Set
    End Property
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>My Company Master Page</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="Form1" runat="server">
        <table cellpadding="3" border="1">
            <tr bgcolor="silver">
                <td colspan="2">
                    <h1><%= PageHeadingTitle %></h1>
                    <b>User's GUID:&nbsp;&nbsp;
                        <asp:Label ID="Label1" runat="server" /></b>
                </td>
            </tr>
            <tr>
                <td>
                    <asp:ContentPlaceHolder ID="ContentPlaceHolder1"
                     runat="server">
                    </asp:ContentPlaceHolder>
                </td>
                <td>
                    <asp:ContentPlaceHolder ID="ContentPlaceHolder2"
                     runat="server">
                    </asp:ContentPlaceHolder>
                </td>
            </tr>
            <tr>
```

```
                <td colspan="2">
                    Copyright 2008 - My Company
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

C#

```
<%@ Master Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            Label1.Text = System.Guid.NewGuid().ToString();
        }
    }

    string m_PageHeadingTitle = "My Company";

    public string PageHeadingTitle
    {
        get
        {
            return m_PageHeadingTitle;
        }
        set
        {
            m_PageHeadingTitle = value;
        }
    }
</script>
```

In this master page example, the master page is exposing the property you created called `PageHeadingTitle`. A default value of `"My Company"` is assigned to this property. You then place it within the HTML of the master page file between some `<h1>` elements. This makes the default value become the heading used on the page within the master page template. Although the master page already has a value it uses for the heading, any content page that is using this master page can override the `<h3>` title heading. The process is shown in Listing 5-11.

Listing 5-11: A content page that overrides the property from the master page

VB

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" %>
<%@ MasterType VirtualPath="~/Wrox.master" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
```

Continued

```
        Master.PageHeadingTitle = "My Company-Division X"
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" MasterPageFile="~/Wrox.master" %>
<%@ MasterType VirtualPath="~/Wrox.master" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Master.PageHeadingTitle = "My Company-Division X";
    }
</script>
```

From the content page, you can assign a value to the property that is exposed from the master page by the use of the `Master` property. As you can see, this is quite simple to do. Remember that not only can you get at any public properties that the master page might expose, but you can also retrieve any methods that the master page contains as well.

The item that makes this all possible is the use of the `MasterType` page directive. The `MasterType` directive allows you to make a strongly typed reference to the master page and allows you to access the master page's properties via the `Master` object.

Earlier, we showed you how to get at the server controls that are on the master page by using the `FindControl()` method. The `FindControl()` method works fine, but it is a late-bound approach, and as such, the method call may fail if the control was removed from markup. Use defensive coding practices and always check for null when returning objects from `FindControl()`. Using the mechanics just illustrated (with the use of public properties shown in Listing 5-10), you can use another approach to expose any server controls on the master page. You may find this approach to be more effective.

To do this, you simply expose the server control as a public property, as shown in Listing 5-12.

Listing 5-12: Exposing a server control from a master page as a public property

VB

```
<%@ Master Language="VB" %>

<script runat="server">
    Public Property MasterPageLabel1() As Label
        Get
            Return Label1
        End Get
        Set(ByVal Value As Label)
            Label1 = Value
        End Set
    End Property
</script>
```

C#

```
<%@ Master Language="C#" %>

<script runat="server">
    public Label MasterPageLabel1
    {
        get
        {
            return Label1;
        }
        set
        {
            Label1 = value;
        }
    }
</script>
```

In this case, a public property called `MasterPageLabel1` provides access to the Label control that uses the `ID` of `Label1`. You can now create an instance of the `MasterPageLabel1` property on the content page and override any of the attributes of the Label server control. So if you want to increase the size of the GUID that the master page creates and displays in the `Label1` server control, you can simply override the `Font.Size` attribute of the Label control, as shown in Listing 5-13.

Listing 5-13: Overriding an attribute from the Label control that is on the master page

VB

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" %>
<%@ MasterType VirtualPath="~/Wrox.master" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Master.MasterPageLabel1.Font.Size = 25
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" MasterPageFile="~/Wrox.master" %>
<%@ MasterType VirtualPath="~/Wrox.master" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Master.MasterPageLabel1.Font.Size = 25;
    }
</script>
```

This approach may be the most effective way to get at any server controls that the master page exposes to the content pages.

Specifying Default Content in the Master Page

As you have seen, the master page enables you to specify content areas that the content page can use. Master pages can consist of just one content area, or they can be made up of multiple content areas. The nice thing about content areas is that when you create a master page, you can specify default content for the content area. This default content can then be left in place and utilized by the content page if you choose not to override it. Listing 5-14 shows a master page that specifies some default content within a content area.

Listing 5-14: Specifying default content in the master page

```
<%@ Master Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>My Company</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
        <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
        Here is some default content.
        </asp:ContentPlaceHolder><p>
        <asp:ContentPlaceHolder ID="ContentPlaceHolder2" runat="server">
        Here is some more default content.
        </asp:ContentPlaceHolder></p>
    </form>
</body>
</html>
```

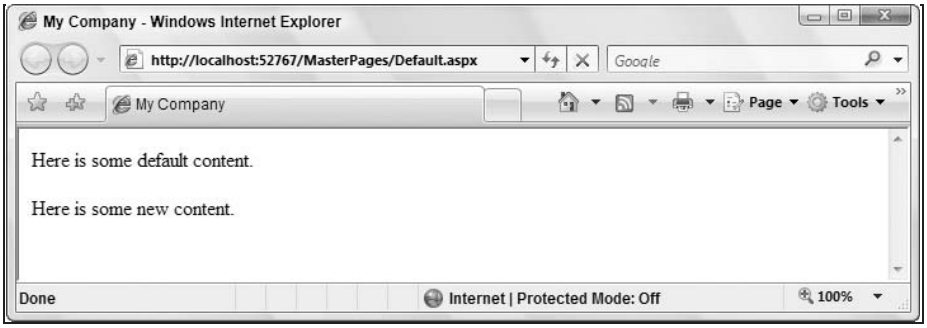
To place default content within one of the content areas of the master page, you simply put it in the `ContentPlaceHolder` server control on the master page itself. Any content page that inherits this master page also inherits the default content. Listing 5-15 shows a content page that overrides just one of the content areas from this master page.

Listing 5-15: Overriding some default content in the content page

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" %>

<asp:Content ID="Content3" ContentPlaceHolderId="ContentPlaceHolder2"
 runat="server">
   Here is some new content.
</asp:Content>
```

This code creates a page with one content area that shows content coming from the master page itself, in addition to other content that comes from the content page (see Figure 5-10).

The other interesting point when you work with content areas in the design mode of Visual Studio 2008 is that the smart tag allows you to work easily with the default content.

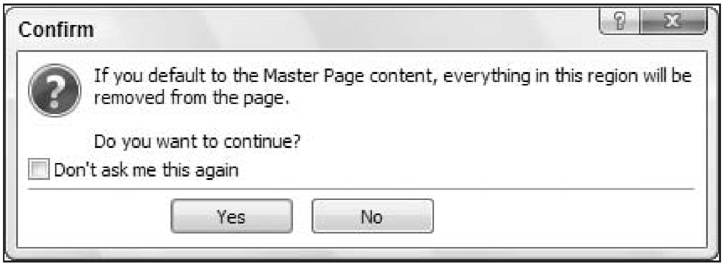

My Company - Windows Internet Explorer
http://localhost:52767/MasterPages/Default.aspx
Google
My Company
Page
Tools
Done
Internet | Protected Mode: Off
100%

Here is some default content.

Here is some new content.

Figure 5-10

When you first start working with the content page, you will notice that all the default content is at first populated in all the Content server controls. You can change the content by clicking on the control's smart tag and selecting the Create Custom Content option from the provided menu. This option enables you to override the master page content and insert your own defined content. After you have placed some custom content inside the content area, the smart tag shows a different option — Default to Master's Content. This option enables you to return the default content that the master page exposes to the content area and to erase whatever content you have already placed in the content area — thereby simply returning to the default content. If you choose this option, you will be warned that you are about to delete any custom content you placed within the Content server control (see Figure 5-11).



If you default to the Master Page content, everything in this region will be removed from the page.

Do you want to continue?

☐ Don't ask me this again

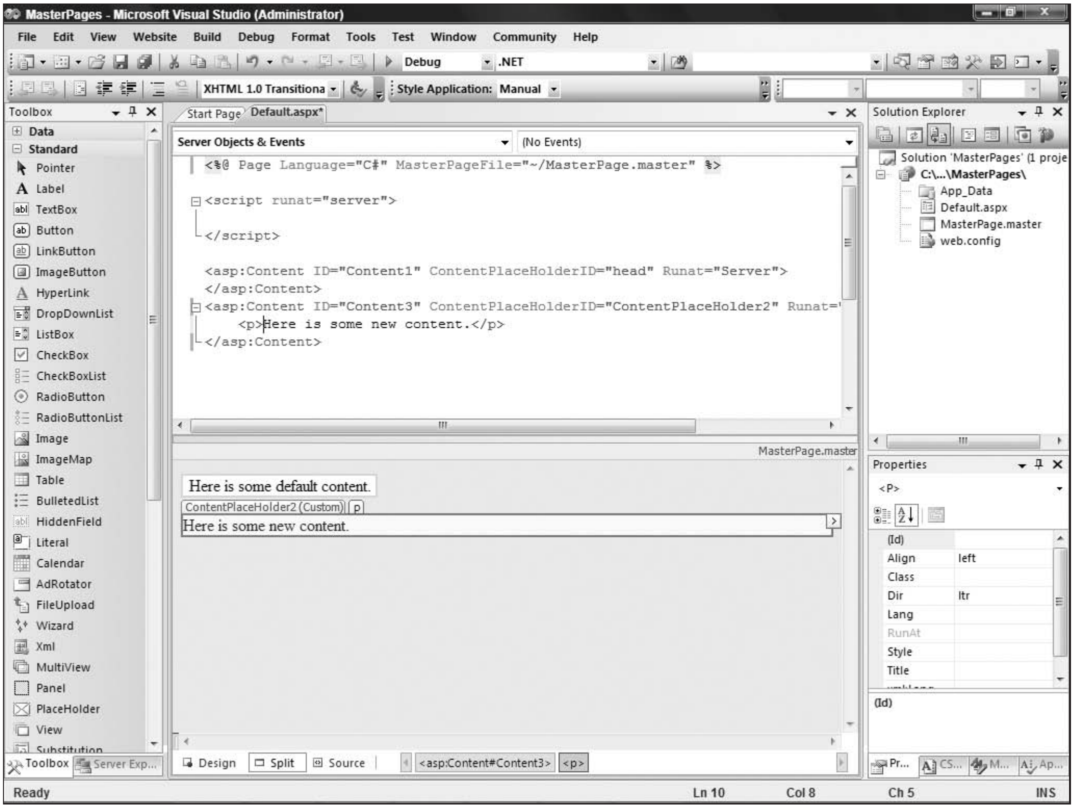
Figure 5-11

After changing one of the Content control's default content, you might be presented with something like Figure 5-12.

Programmatically Assigning the Master Page

From any content page, you can easily assign a master page programmatically. You assign the master page to the content page using the `Page.MasterPageFile` property. This can be used regardless of whether another master page is already assigned in the `@Page` directive.

To accomplish this, you use this property through the `PreInit` event. The `PreInit` event is the earliest point in which you can access the Page lifecycle. For this reason, this is where you need to assign any master page that is used by any content pages. The `PreInit` is an important event to make note of when you are working with master pages, as this is the only point where you can affect both the master and content page before they are combined into a single instance. Listing 5-16 illustrates how to assign the master page programmatically from the content page.



MasterPages - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Format Tools Test Window Community Help
Debug .NET
XHTML 1.0 Transitiona Style Application: Manual
Toolbox
Data
Standard
Pointer
Label
TextBox
Button
LinkButton
ImageButton
HyperLink
DropDownList
ListBox
CheckBox
CheckBoxList
RadioButton
RadioButtonList
Image
ImageMap
Table
BulletedList
HiddenField
Literal
Calendar
AdRotator
FileUpload
Wizard
Xml
MultiView
Panel
PlaceHolder
View
Substitution
Toolbox Server Exp...
Start Page Default.aspx*
Server Objects & Events (No Events)
MasterPage.master
ContentPlaceHolder2 (Custom) p
Design Split Source <asp:Content#Content3> <p>
Solution Explorer
Solution 'MasterPages' (1 proje
C:\...\MasterPages\
App_Data
Default.aspx
MasterPage.master
web.config
Properties
<P>
(Id)
Align left
Class
Dir ltr
Lang
RunAt
Style
Title
(Id)
Pr... CS... M... Ap...
Ready Ln 10 Col 8 Ch 5 INS

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master" %>

<script runat="server">

</script>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="ContentPlaceHolder2" Runat="
    <p>Here is some new content.</p>
</asp:Content>
```

Here is some default content.

Here is some new content.

Figure 5-12

Listing 5-16: Using Page_PreInit to assign the master page programmatically

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs)
        Page.MasterPageFile = "~/MyMasterPage.master"
    End Sub
</script>
```

C#

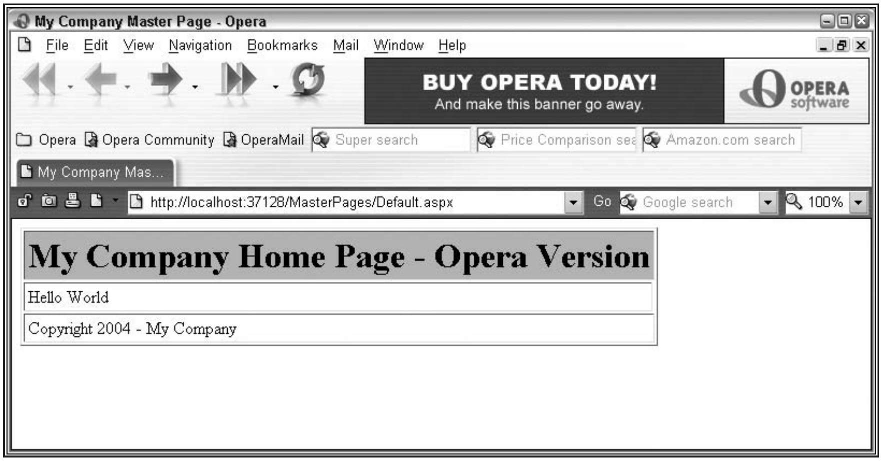
```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_PreInit(object sender, EventArgs e)
    {
        Page.MasterPageFile = "~/MyMasterPage.master";
    }
</script>
```

In this case, when the page is dynamically being generated, the master page is assigned to the content page in the beginning of the page construction process. It is important to note that the content page must have the expected Content controls; otherwise an error is thrown.

Nesting Master Pages

I hope you see the power that master pages provide to help you create templated Web applications. So far, you have been creating a single master page that the content page can use. Most companies and organizations, however, are not just two layers. Many divisions and groups exist within the organization that might want to use variations of the master by, in effect, having a master page within a master page. With ASP.NET, this is quite possible.

For example, imagine that Reuters is creating a master page to be used throughout the entire company intranet. Not only does the Reuters enterprise want to implement this master page company-wide, but various divisions within Reuters also want to provide templates for the subsections of the intranet directly under their control. Reuters Europe and Reuters America, for example, each wants its own unique master page, as illustrated in Figure 5-13.


My Company Master Page - Opera
File Edit View Navigation Bookmarks Mail Window Help
BUY OPERA TODAY!
And make this banner go away.
OPERA software
Opera
Opera Community
OperaMail
Super search
Price Comparison sea
Amazon.com search
My Company Mas...
Go
Google search
100%

http://localhost:37128/MasterPages/Default.aspx

My Company Home Page - Opera Version

Hello World

Copyright 2004 - My Company

Figure 5-13

To do this, the creators of the Reuters Europe and Reuters America master pages simply create a master page that inherits from the global master page, as illustrated in Listing 5-17.

Listing 5-17: The main master page

ReutersMain.master

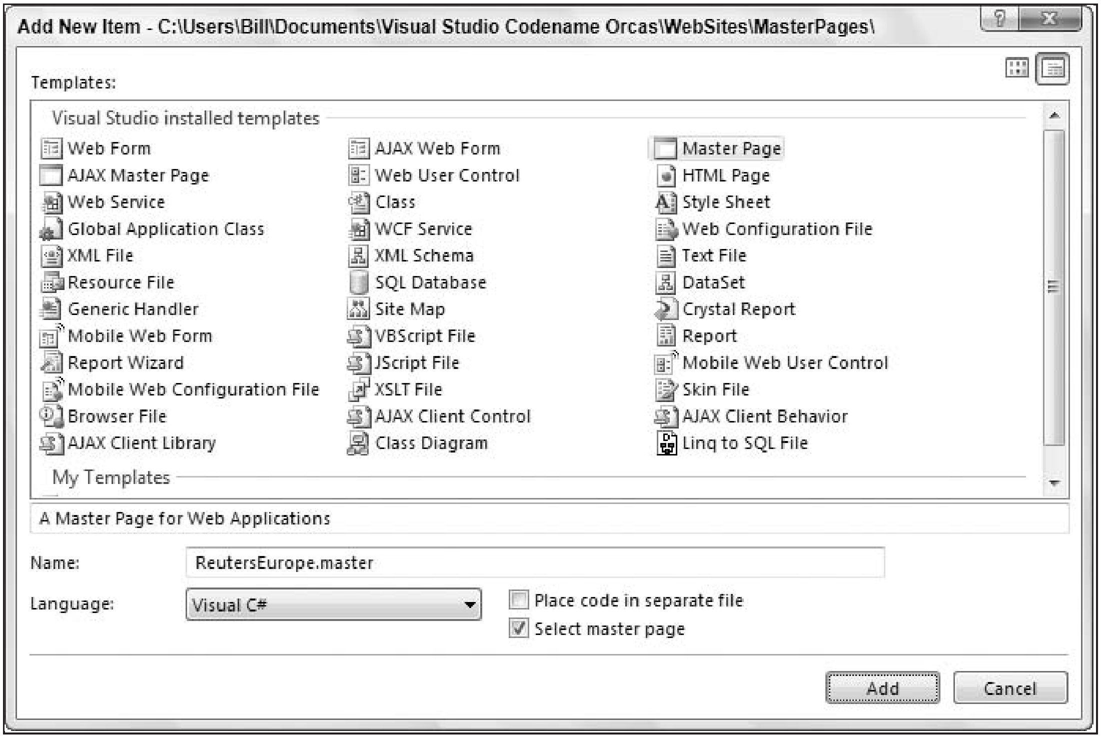
```
<%@ Master Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Reuters</title>
```

Continued

```
        <asp:ContentPlaceHolder id="head" runat="server">
        </asp:ContentPlaceHolder>
    </head>
    <body>
        <form id="form1" runat="server">
            <p><asp:Label ID="Label1" runat="server" BackColor="LightGray"
                BorderColor="Black" BorderWidth="1px" BorderStyle="Solid"
                Font-Size="XX-Large"> Reuters Main Master Page </asp:Label></p>
            <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
            </asp:ContentPlaceHolder>
        </form>
    </body>
    </html>
```

This is a simple master page, but excellent for showing you how this nesting capability works. The main master page is the master page used globally in the company. It has the ContentPlaceHolder server control with the `ID` of `ContentPlaceHolder1`.

When you create a submaster or nested master page, you accomplish this task in the same manner as you would when building any other master page. From the Add New Item dialog, select the Master Page option and make sure you have the Select master page option selected, as illustrated in Figure 5-14. This will take you again to the dialog that will allow you to make a master page selection.


Add New Item - C:\Users\Bill\Documents\Visual Studio Codename Orcas\WebSites\MasterPages\

Templates: Master Page

A Master Page for Web Applications

Name: ReutersEurope.master

Language: Visual C#

☐ Place code in separate file

☑ Select master page

Templates:
Visual Studio installed templates
Web Form
AJAX Master Page
Web Service
Global Application Class
XML File
Resource File
Generic Handler
Mobile Web Form
Report Wizard
Mobile Web Configuration File
Browser File
AJAX Client Library
AJAX Web Form
Web User Control
Class
WCF Service
XML Schema
SQL Database
Site Map
VBScript File
JScript File
XSLT File
AJAX Client Control
Class Diagram
Master Page
HTML Page
Style Sheet
Web Configuration File
Text File
DataSet
Crystal Report
Report
Mobile Web User Control
Skin File
AJAX Client Behavior
Linq to SQL File
My Templates
Add
Cancel

Figure 5-14

Listing 5-18 illustrates how you can work with this main master from a submaster file.

Listing 5-18: The submaster page

ReutersEurope.master

```
<%@ Master Language="VB" MasterPageFile="~/ReutersMain.master" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderId="ContentPlaceHolder1"
 runat="server">
    <asp:Label ID="Label1" runat="server" BackColor="#E0E0E0" BorderColor="Black"
     BorderStyle="Dotted" BorderWidth="2px" Font-Size="Large">
     Reuters Europe </asp:Label><br /><hr />

        <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
        </asp:ContentPlaceHolder>
</asp:Content>
```

Looking this page over, you can see that it isn't much different than a typical `.aspx` page that makes use of a master page. The `MasterPageFile` attribute is used just the same, but instead of using the `@Page` directive, the `@Master` page directive is used. Then the `Content2` control also uses the `ContentPlaceHolderId` attribute of the Content control. This attribute is tying this content area to the content area `ContentPlaceHolder1`, which is defined in the main master page.

One new feature of ASP.NET 3.5 is the ability to view nested master pages directly in the Design view of Visual Studio 2008. The previous Visual Studio 2005 would actually throw an error when trying to present a nested master page. Figure 5-15 shows a nested master page in the Design view of Visual Studio 2008.

Within the submaster page presented in Listing 5-18, you can also now use as many ContentPlaceHolder server controls as you want. Any content page that uses this master can use these controls. Listing 5-19 shows a content page that uses the submaster page `ReutersEurope.master`.

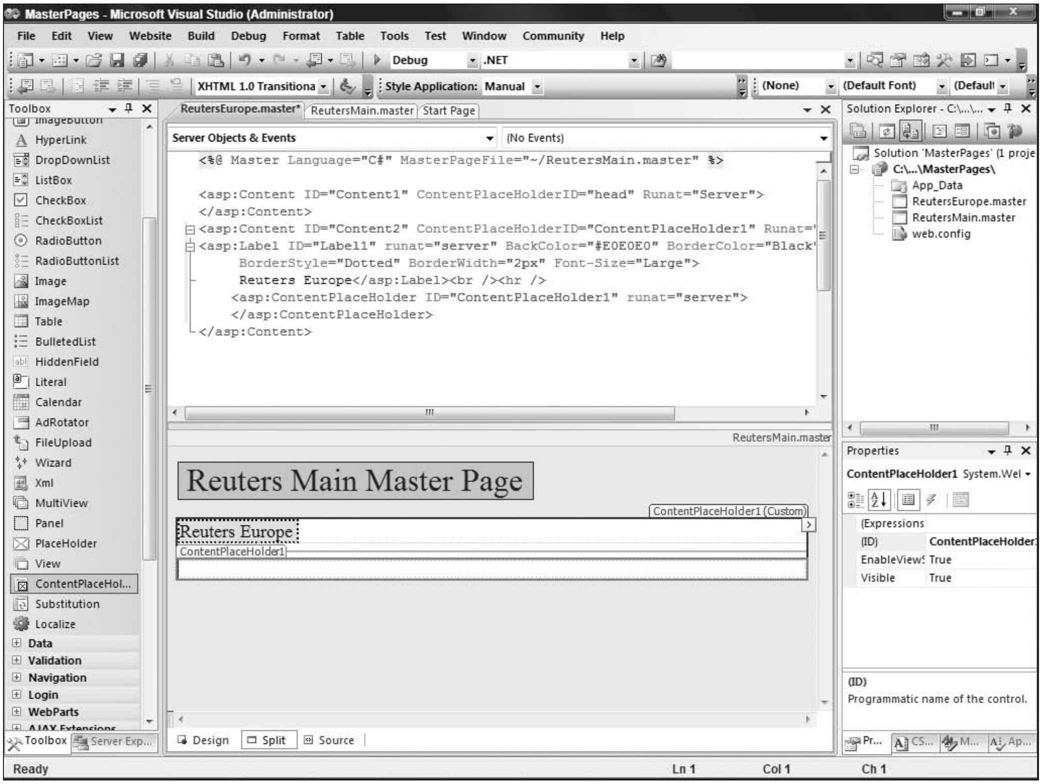
Listing 5-19: The content page

Default.aspx

```
<%@ Page Language="VB" MasterPageFile="~/ReutersEurope.master" %>

<asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceHolder1"
 runat="server">
    Hello World
</asp:Content>
```

As you can see, in this content page the value of the `MasterPageFile` attribute in the `Page` directive is the submaster page that you created. Inheriting the `ReutersEurope` master page actually combines both master pages (`ReutersMain.master` and `ReutersEurope.master`) into a single master page.

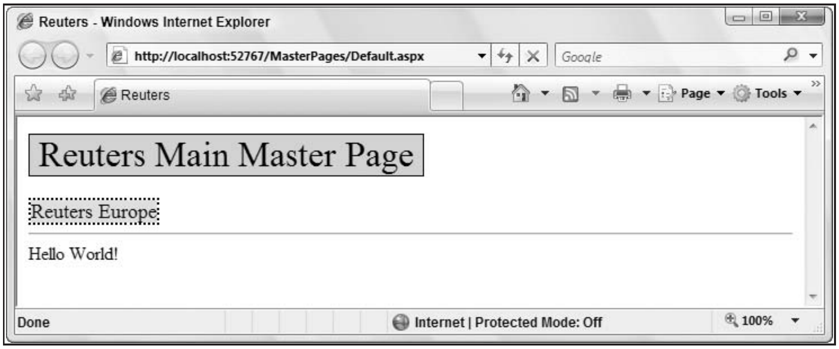
The Content control in this content page points to the content area defined in the submaster page as well. You can see this in the code with the use of the `ContentPlaceHolderId` attribute. In the end, you get a very non-artistic page, as shown in Figure 5-16.



```
<%@ Master Language="C#" MasterPageFile="~/ReutersMain.master" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat='
<asp:Label ID="Label1" runat="server" BackColor="#E0E0E0" BorderColor="Black'
     BorderStyle="Dotted" BorderWidth="2px" Font-Size="Large">
     Reuters Europe</asp:Label><br /><hr />
    <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
    </asp:ContentPlaceHolder>
</asp:Content>
```

MasterPages - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Format Table Tools Test Window Community Help
Debug
.NET
XHTML 1.0 Transitiona
Style Application: Manual
(None)
(Default Font)
Toolbox
HyperLink
DropDownList
ListBox
CheckBox
CheckBoxList
RadioButton
RadioButtonList
Image
ImageMap
Table
BulletedList
HiddenField
Literal
Calendar
AdRotator
FileUpload
Wizard
Xml
MultiView
Panel
PlaceHolder
View
ContentPlaceHol...
Substitution
Localize
Data
Validation
Navigation
Login
WebParts
Toolbox
Server Exp...
ReutersEurope.master*
ReutersMain.master
Start Page
Server Objects & Events
(No Events)
ReutersMain.master
Reuters Main Master Page
ContentPlaceHolder1 (Custom)
Reuters Europe
ContentPlaceHolder1
Design
Split
Source
Solution Explorer - C:\...\...
Solution 'MasterPages' (1 proje
C:\...\MasterPages\
App_Data
ReutersEurope.master
ReutersMain.master
web.config
Properties
ContentPlaceHolder1 System.Wel
(Expressions)
(ID) ContentPlaceHolder
EnableView True
Visible True
(ID)
Programmatic name of the control.
Ready
Ln 1
Col 1
Ch 1

Figure 5-15



Reuters - Windows Internet Explorer
http://localhost:52767/MasterPages/Default.aspx
Google
Reuters
Page
Tools
Reuters Main Master Page
Reuters Europe
Hello World!
Done
Internet | Protected Mode: Off
100%

Figure 5-16

As you can see, creating a content page that uses a submaster page works pretty well.

Container-Specific Master Pages

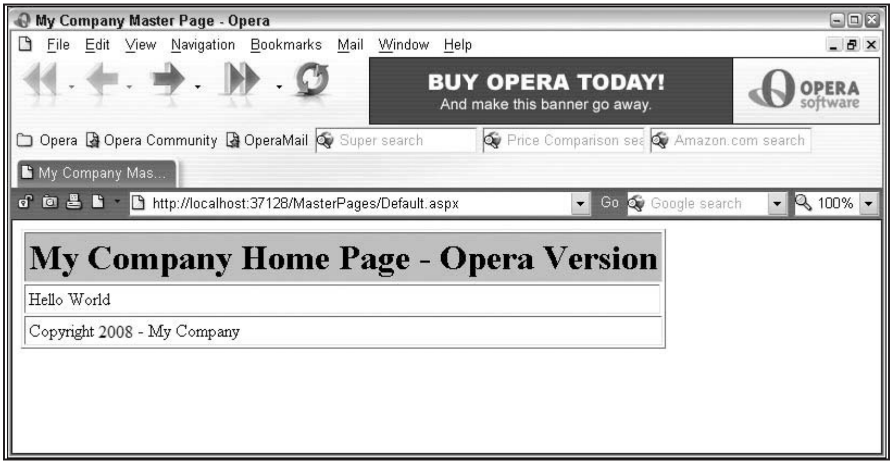
In many cases, developers are building applications that will be viewed in a multitude of different containers. Some viewers may view the application in Microsoft Internet Explorer and some might view it using Opera or Netscape Navigator. And still other viewers may call up the application on a Pocket PC or Nokia cell phone.

For this reason, ASP.NET allows you to use multiple master pages within your content page. Depending on the viewing container used by the end user, the ASP.NET engine pulls the appropriate master file. Therefore, you want to build container-specific master pages to provide your end users with the best possible viewing experience by taking advantage of the features that a specific container provides. The capability to use multiple master pages is demonstrated in Listing 5-20.

Listing 5-20: A content page that can work with more than one master page

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master"
       Mozilla:MasterPageFile="~/WroxMozilla.master"
       Opera:MasterPageFile="~/WroxOpera.master" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderId="ContentPlaceHolder1"
 runat="server">
    Hello World
</asp:Content>
```

As you can see from this example content page, it can work with three different master page files. The first one uses the attribute `MasterPageFile`. This is the default setting used for any page that doesn't fit the criteria for any of the other options. This means that if the requestor is not a Mozilla or Opera browser, the default master page, `Wrox.master`, is used. However, if the requestor is an Opera browser, `WroxOpera.master` is used instead. This is illustrated in Figure 5-17.


My Company Master Page - Opera
File Edit View Navigation Bookmarks Mail Window Help
BUY OPERA TODAY!
And make this banner go away.
OPERA software
Opera
Opera Community
OperaMail
Super search
Price Comparison sea
Amazon.com search
My Company Mas...
http://localhost:37128/MasterPages/Default.aspx
Go
Google search
100%

My Company Home Page - Opera Version

Hello World

Copyright 2008 - My Company

Figure 5-17

You can find a list of available browsers on the production server where the application will be hosted at `C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG\Browsers`. Some of the available options include the following:

- avantgo
- cassio
- Default
- docomo
- ericsson
- EZWap
- gateway
- generic
- goAmerica
- ie
- Jataayu
- jphone
- legend
- MME
- mozilla
- netscape
- nokia
- openwave
- opera
- palm
- panasonic
- pie
- webtv
- winwap
- xiino

Of course, you can also add any additional `.browser` files that you deem necessary.

Event Ordering

When you work with master pages and content pages, both can use the same events (such as the `Load` event). Be sure you know which events come before others. You are bringing two classes together to create a single page class, and a specific order is required. When an end user requests a content page in the browser, the event ordering is as follows:

- **Master page child controls initialization::** All server controls contained within the master page are first initialized.
- **Content page child controls initialization::** All server controls contained in the content page are initialized.
- **Master page initialization::** The master page itself is initialized.
- **Content page initialization::** The content page is initialized.
- **Content page load::** The content page is loaded (this is the `Page_Load` event followed by the `Page_LoadComplete` event).
- **Master page load::** The master page is loaded (this is also the `Page_Load` event followed by the `Page_LoadComplete` event).
- **Master page child controls load::** The server controls on the master page are loaded onto the page.
- **Content page child controls load::** The server controls on the content page are loaded onto the page.

Pay attention to this event ordering when building your applications. If you want to use server control values that are contained on the master page within a specific content page, for example, you can't retrieve the values of these server controls from within the content page's `Page_Load` event. This is because this event is triggered before the master page's `Page_Load` event. This problem prompted the creation of the new `Page_LoadComplete` event. The content page's `Page_LoadComplete` event follows the master page's `Page_Load` event. You can, therefore, use this ordering to get at values from the master page even though it isn't populated when the content page's `Page_Load` event is triggered.

Caching with Master Pages

When working with typical `.aspx` pages, you can apply output caching to the page by using the following construct (or variation thereof):

```
<%@ OutputCache Duration="10" Varybyparam="None" %>
```

This caches the page in the server's memory for 10 seconds. Many developers use output caching to increase the performance of their ASP.NET pages. It also makes a lot of sense for use on pages with data that doesn't become stale too quickly.

How do you go about applying output caching to ASP.NET pages when working with master pages? First, you cannot apply caching to just the master page. You cannot put the `OutputCache` directive on the master page itself. If you do so, on the page's second retrieval, you get an error because the application cannot find the cached page.

To work with output caching when using a master page, stick the `OutputCache` directive in the content page. This caches both the contents of the content page as well as the contents of the master page (remember, it is just a single page at this point). The `OutputCache` directive placed in the master page does not cause the master page to produce an error, but it will not be cached. This directive works in the content page only.

ASP.NET AJAX and Master Pages

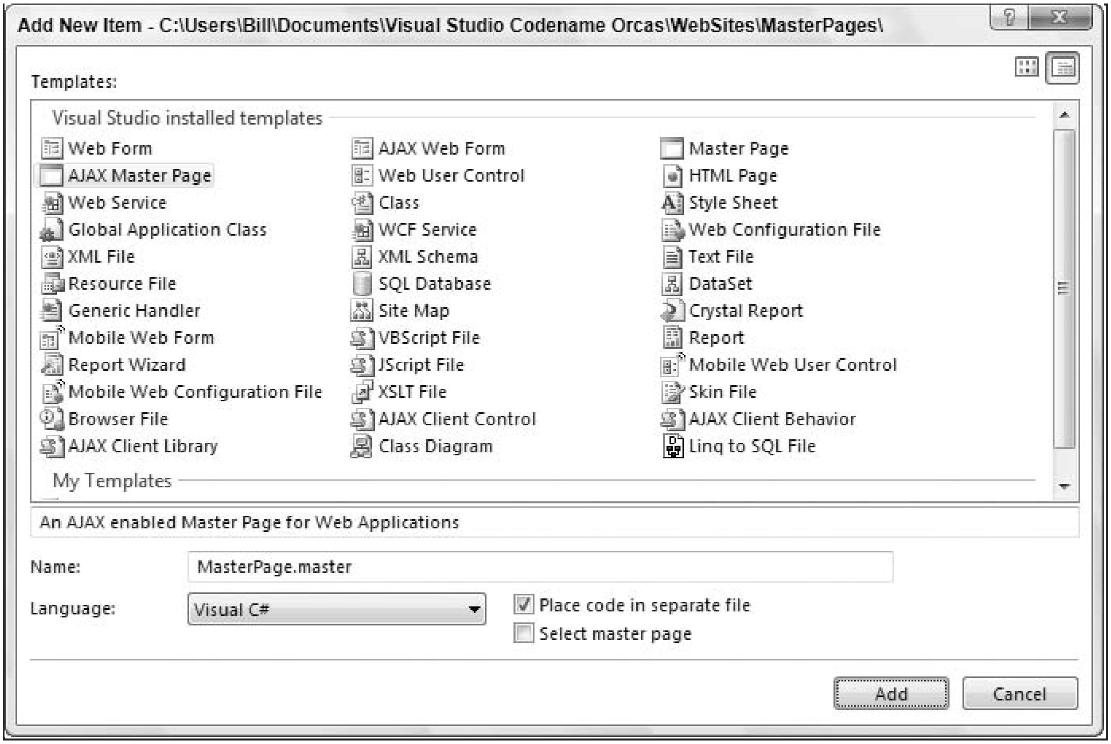
Many of the larger ASP.NET applications today make use of master pages and the power this technology provides in the ability of building templated Web sites. ASP.NET 3.5 introduces ASP.NET AJAX as part of the default install and you will find that master pages and Ajax go together quite well.

ASP.NET AJAX is covered in Chapter 19 of this book.

Every page that is going to make use of AJAX capabilities will have to have the new ScriptManager control on the page. If the page that you want to use AJAX with is a content page making use of a master page, then you are going to have to place the ScriptManager control on the master page itself.

Note that you can only have one ScriptManager on a page.

ASP.NET 3.5 makes this process easy. Opening up the Add New Item dialog, you will notice that in addition to the Master Page option, you will also find an AJAX Master Page option as illustrated in Figure 5-18.


Add New Item - C:\Users\Bill\Documents\Visual Studio Codename Orcas\WebSites\MasterPages\
Templates:
Visual Studio installed templates
Web Form
AJAX Master Page
Web Service
Global Application Class
XML File
Resource File
Generic Handler
Mobile Web Form
Report Wizard
Mobile Web Configuration File
Browser File
AJAX Client Library
AJAX Web Form
Web User Control
Class
WCF Service
XML Schema
SQL Database
Site Map
VBScript File
JScript File
XSLT File
AJAX Client Control
Class Diagram
Master Page
HTML Page
Style Sheet
Web Configuration File
Text File
DataSet
Crystal Report
Report
Mobile Web User Control
Skin File
AJAX Client Behavior
Linq to SQL File
My Templates
Add
Cancel

An AJAX enabled Master Page for Web Applications

Name: MasterPage.master

Language: Visual C#

☑ Place code in separate file

☐ Select master page

Figure 5-18

Selecting this option produces a page like the one presented in Listing 5-21.

Listing 5-21: The AJAX master page

```
<%@ Master Language="VB" %>

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server" />
        <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
```

```
            </asp:ContentPlaceHolder>
        </div>
        </form>
    </body>
    </html>
```

As you can see from Listing 5-21, the only real difference between this AJAX master page and the standard master page is the inclusion of the ScriptManager server control. You are going to want to use this technique if your master page includes any AJAX capabilities whatsoever, even if the content page makes no use of AJAX at all.

The ScriptManager control on the master page also is beneficial if you have common JavaScript items to place on all the pages of your Web application. For instance, Listing 5-22 shows how you could easily include JavaScript on each page through the master page.

Listing 5-22: Including scripts through your master page

```
<%@ Master Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
            <Scripts>
                <asp:ScriptReference Path="myScript.js" />
            </Scripts>
        </asp:ScriptManager>
        <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">

        </asp:ContentPlaceHolder>
    </div>
    </form>
</body>
</html>
```

In this example, the `myScript.js` file will now be included on every content page that makes use of this AJAX master page. If your content page also needs to make use of Ajax capabilities, then you simply cannot add another ScriptManager control to the page. Instead, the content page will need to make use of the ScriptManager control that is already present on the master page.

That said, if your content page needs to add additional items to the ScriptManager control, it is able to access this control on the master page using the ScriptManagerProxy server control. Using the ScriptManagerProxy control gives you the ability to add any items to the ScriptManager that are completely specific to the instance of the content page that makes the inclusions.

For instance, Listing 5-23 shows how a content page would add additional scripts to the page through the ScriptManagerProxy control.

Listing 5-23: Adding additional items using the ScriptManagerProxy control

```
<%@ Page Language="VB" MasterPageFile="~/AjaxMaster.master" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1"
 Runat="Server">
 <asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">
     <Scripts>
         <asp:ScriptReference Path="myOtherScript.js" />
     </Scripts>
 </asp:ScriptManagerProxy>
</asp:Content>
```

In this case, this content page is using the ScriptManagerProxy control to add an additional script to the page. This ScriptManagerProxy control works exactly the same as the main ScriptManager control except that it is meant for content pages making use of a master page. The ScriptManagerProxy control will then interact with the page's ScriptManager control to perform the actions necessary.

Summary

When you create applications that use a common header, footer, or navigation section on pretty much every page of the application, master pages are a great solution. Master pages are easy to implement and enable you to make changes to each and every page of your application by changing a single file. Imagine how much easier this makes managing large applications that contain thousands of pages.

This chapter described master pages in ASP.NET and explained how you build and use master pages within your Web applications. In addition to the basics, the chapter covered master page event ordering, caching, and specific master pages for specific containers. In the end, when you are working with templated applications, master pages should be your first option — the power of this approach is immense.

6

Themes and Skins

When you build a Web application, it usually has a similar look-and-feel across all its pages. Not too many applications are designed with each page dramatically different from the next. Generally, for your applications, you use similar fonts, colors, and server control styles across all the pages.

You can apply these common styles individually to each and every server control or object on each page, or you can use a capability provided by ASP.NET 3.5 to centrally specify these styles. All pages or parts of pages in the application can then access them.

Themes are the text-based style definitions in ASP.NET 3.5 that are the focus of this chapter.

Using ASP.NET Themes

Themes are similar to Cascading Style Sheets (CSS) in that they enable you to define visual styles for your Web pages. Themes go further than CSS, however, in that they allow you to apply styles, graphics, and even CSS files themselves to the pages of your applications. You can apply ASP.NET themes at the application, page, or server control level.

Applying a Theme to a Single ASP.NET Page

In order to see how to use one of these themes, create a basic page, which includes some text, a text box, a button, and a calendar, as shown in Listing 6-1.

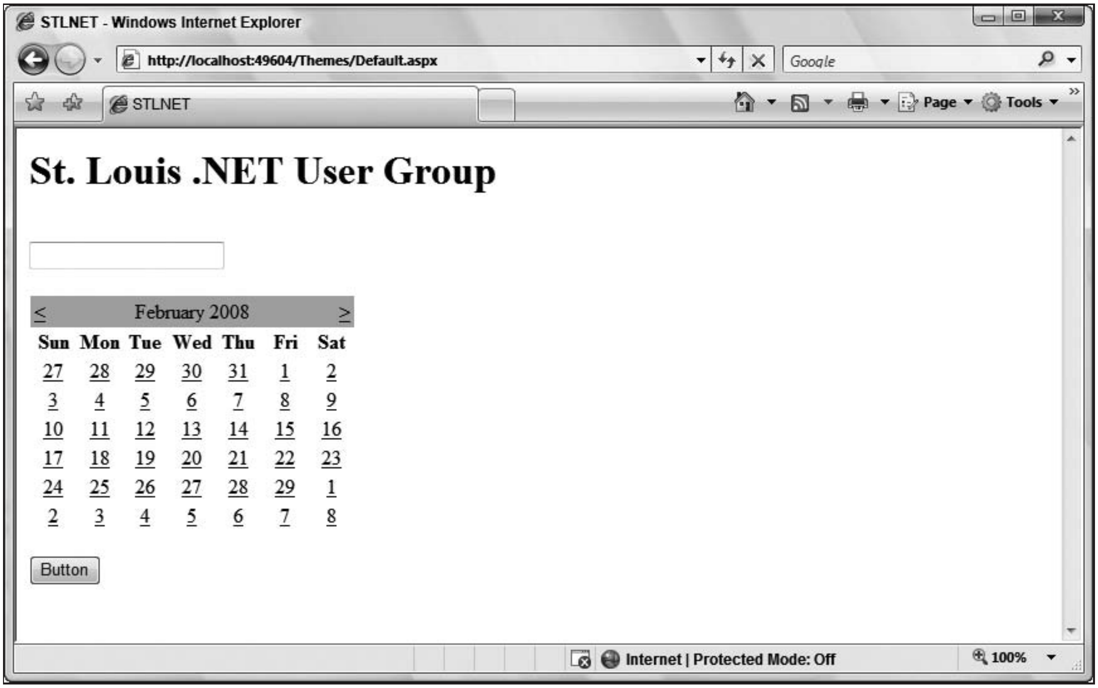
Listing 6-1: An ASP.NET page that does not use themes

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>STLNET</title>
```

Continued

```
</head>
<body>
    <form id="form1" runat="server">
        <h1>St. Louis .NET User Group</h1><br />
        <asp:Textbox ID="TextBox1" runat="server" /><br />
        <br />
        <asp:Calendar ID="Calendar1" runat="server" /><br />
        <asp:Button ID="Button1" runat="server" Text="Button" />
    </form>
</body>
</html>
```

This simple page shows some default server controls that appear just as you would expect, but that you can change with one of these new ASP.NET themes. When this theme-less page is called in the browser, it should look like Figure 6-1.



STLNET - Windows Internet Explorer
http://localhost:49604/Themes/Default.aspx
Google
STLNET
Page
Tools
Button
Internet | Protected Mode: Off
100%

St. Louis .NET User Group

<			February 2008			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	1
2	3	4	5	6	7	8

Figure 6-1

You can instantly change the appearance of this page without changing the style of each server control on the page. From within the `Page` directive, you simply apply an ASP.NET theme that you have either built (shown later in this chapter) or downloaded from the Internet:

```
<%@ Page Language="VB" Theme="SmokeAndGlass" %>
```

Adding the `Theme` attribute to the `Page` directive changes the appearance of everything on the page that is defined in an example `SmokeAndGlass` theme file. Using this theme, when we invoked the page in the browser, we got the result shown in Figure 6-2.

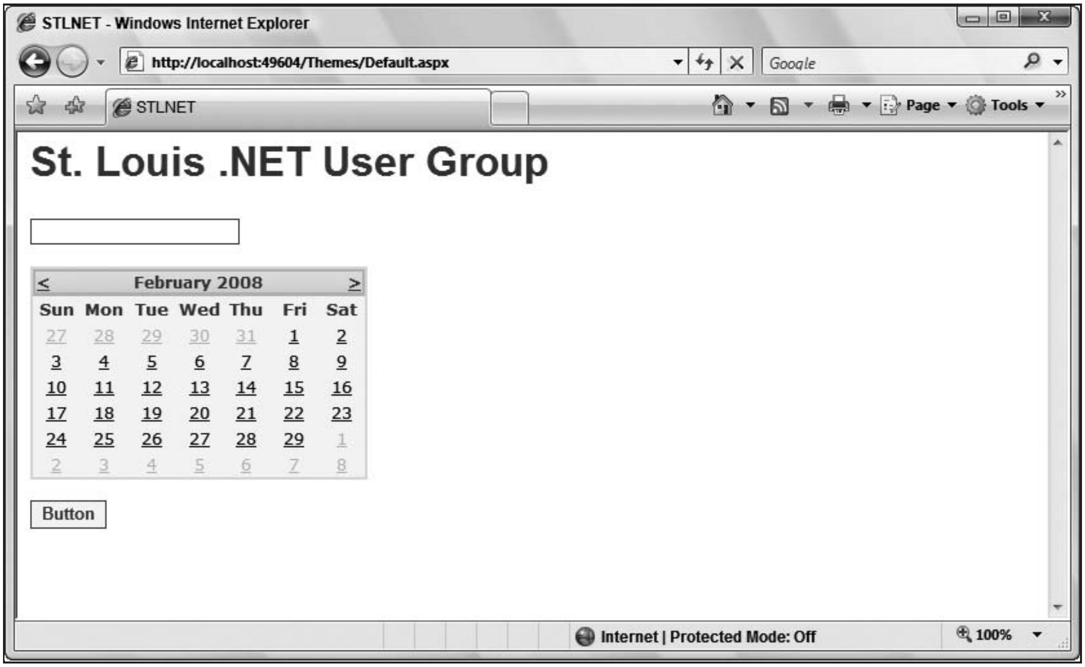

STLNET - Windows Internet Explorer
http://localhost:49604/Themes/Default.aspx
Google
STLNET
Page
Tools
St. Louis .NET User Group
February 2008
Sun Mon Tue Wed Thu Fri Sat
Button
Internet | Protected Mode: Off
100%

Figure 6-2

From here, you can see that everything — including the font, font color, text box, button, and more — has changed appearance. If you have multiple pages, you may find that it is nice not to have to think about applying styles to everything you do as you build because the styles are already centrally defined for you.

Applying a Theme to an Entire Application

In addition to applying an ASP.NET theme to your ASP.NET pages using the `Theme` attribute within the `Page` directive, you can also apply it at an application level from the `web.config` file. This is illustrated in Listing 6-2.

Listing 6-2: Applying a theme application-wide from the web.config file

```
<?xml version="1.0"?>

<configuration>
   <system.web>
      <pages theme="SmokeAndGlass" />
   </system.web>
</configuration>
```

If you specify the theme in the `web.config` file, you do not need to define the theme again in the `Page` directive of your ASP.NET pages. This theme is applied automatically to each and every page within your application. If you wanted to apply the theme to only a specific part of the application in this fashion, then you can do the same, but in addition, make use of the <location/> element to specify the areas of the applications for which the theme should be applied.

Removing Themes from Server Controls

Whether themes are set at the application level or on a page, at times you want an alternative to the theme that has been defined. For example, change the text box server control that you have been working with (from Listing 6-1) by making its background black and using white text:

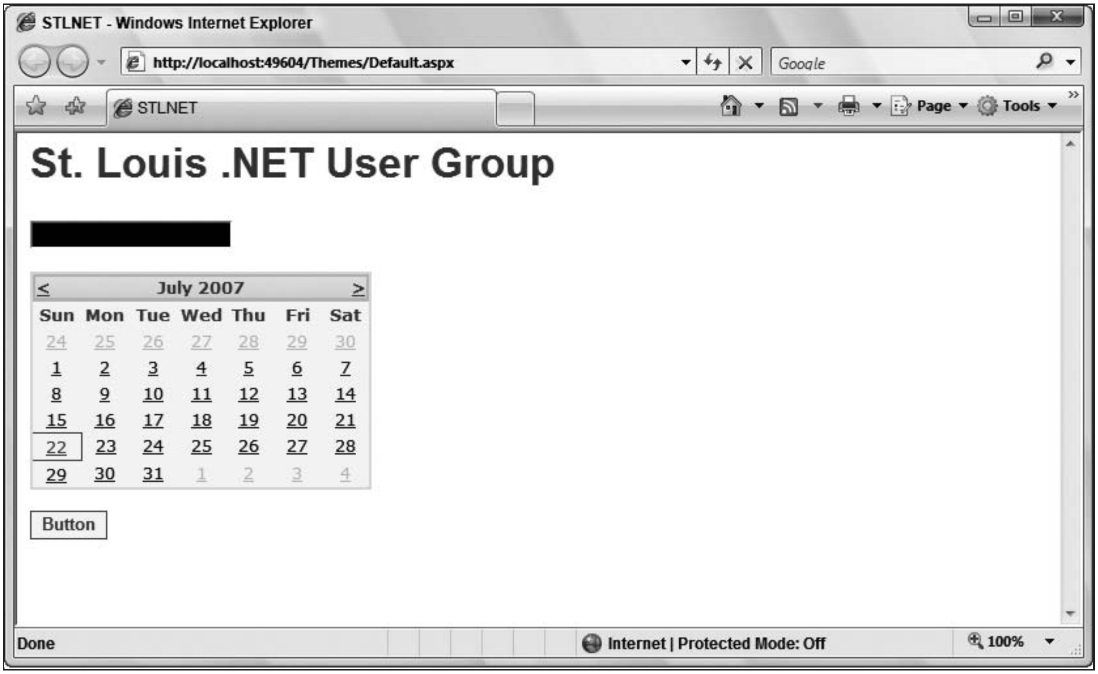
```
<asp:Textbox ID="TextBox1" runat="server"
 BackColor="#000000" ForeColor="#ffffff" />
```

The black background color and the color of the text in the text box are specified directly in the control itself with the use of the `BackColor` and `ForeColor` attributes. If you have applied a theme to the page where this text box control is located, however, you will not see this black background or white text because these changes are overridden by the theme itself.

To apply a theme to your ASP.NET page but not to this text box control, you simply use the `EnableTheming` property of the text box server control:

```
<asp:Textbox ID="TextBox1" runat="server"
 BackColor="#000000" ForeColor="#ffffff" EnableTheming="false" />
```

If you apply this property to the text box server control from Listing 6-1 while the `SmokeAndGlass` theme is still applied to the entire page, the theme is applied to every control on the page *except* the text box. This result is shown in Figure 6-3.


STLNET - Windows Internet Explorer
http://localhost:49604/Themes/Default.aspx
Google
STLNET
Page
Tools
St. Louis .NET User Group
Button
Done
Internet | Protected Mode: Off
100%

<			July 2007			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

Figure 6-3

If you want to turn off theming for multiple controls within a page, consider using the Panel control (or any container control) to encapsulate a collection of controls and then set the `EnableTheming` attribute of the Panel control to `False`. This disables theming for each control contained within the Panel control.

Removing Themes from Web Pages

Now what if, when you set the theme for an entire application in the `web.config` file, you want to exclude a single ASP.NET page? It is quite possible to remove a theme setting at the page level, just as it is at the server control level.

The `Page` directive includes an `EnableTheming` attribute that can be used to remove theming from your ASP.NET pages. To remove the theme that would be applied by the theme setting in the `web.config`, you simply construct your `Page` directive in the following manner:

```
<%@ Page Language="VB" EnableTheming="False" %>
```

This construct sets the theme to nothing — thereby removing any settings that were specified in the `web.config` file. When this directive is set to `False` at the page or control level, the `Theme` directory is not searched, and no `.skin` files are applied (`.skin` files are used to define styles for ASP.NET server controls). When it is set to `True` at the page or control level, the `Theme` directory is searched and `.skin` files are applied.

If themes are disabled because the `EnableTheming` attribute is set to `False` at the page level, you can still enable theming for specific controls on this page by setting the `EnableTheming` property for the control to `True` and applying a specific theme at the same time, as illustrated here:

```
<asp:Textbox ID="TextBox1" runat="server"
 BackColor="#000000" ForeColor="#ffffff" EnableTheming="true" SkinID="mySkin" />
```

Understanding Themes When Using Master Pages

When working with ASP.NET applications that make use of master pages, notice that both the `Page` and `Master` page directives include an `EnableTheming` attribute.

Master pages are covered in Chapter 5.

If both the `Page` and `Master` page directives include the `EnableTheming` attribute, what behavior results if both are used? Suppose you have defined your theme in the `web.config` file of your ASP.NET application and you specify in the master page that theming is disabled using the `EnableTheming` attribute as shown here:

```
<%@ Master Language="VB" EnableTheming="false" %>
```

If this is the case, what is the behavior for any content pages using this master page? If the content page that is using this master page does not make any specification on theming (it does not use the `EnableTheming` attribute), what is specified in the master page naturally takes precedence and no theme is utilized as required by the `false` setting. Even if you have set the `EnableTheming` value in the content page, any value that is specified in the master page takes precedence. This means that if theming is set to `false` in the master page and set to `true` in the content page, the page is constructed with the value provided from the master page — in this case, `false`. Even if the value is set to `false` in the master page, however, you can override this setting at the control level rather than doing it in the `Page` directive of the content page.

Understanding the StyleSheetTheme Attribute

The `Page` directive also includes the attribute `StyleSheetTheme` that you can use to apply themes to a page. So, the big question is: If you have a `Theme` attribute and a `StyleSheetTheme` attribute for the `Page` directive, what is the difference between the two?

```
<%@ Page Language="VB" StyleSheetTheme="Summer" %>
```

The `StyleSheetTheme` attribute works the same as the `Theme` attribute in that it can be used to apply a theme to a page. The difference is that the when attributes are set locally on the page within a particular control, the attributes are overridden by the theme if you use the `Theme` attribute. They are kept in place, however, if you apply the page's theme using the `StyleSheetTheme` attribute. Suppose you have a text box control like the following:

```
<asp:Textbox ID="TextBox1" runat="server"
 BackColor="#000000" ForeColor="#ffffff" />
```

In this example, the `BackColor` and `ForeColor` settings are overridden by the theme if you have applied it using the `Theme` attribute in the `Page` directive. If, instead, you applied the theme using the `StyleSheetTheme` attribute in the `Page` directive, the `BackColor` and `ForeColor` settings remain in place, even if they are explicitly defined in the theme.

Creating Your Own Themes

You will find that creating themes in ASP.NET is a rather simple process — although sometimes it does require some artistic capabilities. The themes you create can be applied at the application, page, or server control level. Themes are a great way to easily apply a consistent look-and-feel across your entire application.

Creating the Proper Folder Structure

In order to create your own themes for an application, you first need to create the proper folder structure in your application. To do this, right-click your project and add a new folder. Name the folder `App_Themes`. You can also create this folder by right-clicking on your project in Visual Studio and selecting Add ASP.NET Folder ⇨ Theme. Notice when you do this that the theme folder within the `App_Themes` folder does not have the typical folder icon next to it, but instead has a folder icon that includes a paintbrush. This is shown in Figure 6-4.

Within the `App_Themes` folder, you can create an additional theme folder for each and every theme that you might use in your application. For instance, if you are going to have four themes — *Summer*, *Fall*, *Winter*, and *Spring* — then you create four folders that are named appropriately.

You might use more than one theme in your application for many reasons — season changes, day/night changes, different business units, category of user, or even user preferences.

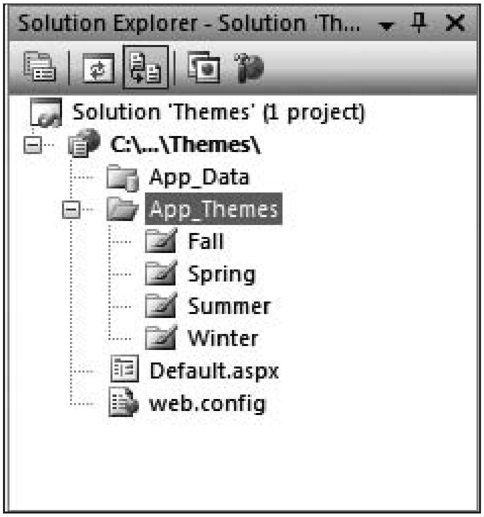

Solution Explorer - Solution 'Th...
Solution 'Themes' (1 project)
C:\...\Themes\
App_Data
App_Themes
Fall
Spring
Summer
Winter
Default.aspx
web.config

Figure 6-4

Each theme folder must contain the elements of the theme, which can include the following:

- ❑ A single skin file
- ❑ CSS files
- ❑ Images

Creating a Skin

A *skin* is a definition of styles applied to the server controls in your ASP.NET page. Skins can work in conjunction with CSS files or images. To create a theme to use in your ASP.NET applications, you use just a single skin file in the theme folder. The skin file can have any name, but it must have a `.skin` file extension.

Even though you have four theme folders in your application, concentrate on the creation of the Summer theme for the purposes of this chapter. Right-click the `Summer` folder, select Add New Item, and select Skin File from the listed options. Name the file **Summer.skin**. Then complete the skin file as shown in Listing 6-3.

Listing 6-3: The Summer.skin file

```
<asp:Label runat="server" ForeColor="#004000" Font-Names="Verdana"
            Font-Size="X-Small" />

<asp:Textbox runat="server" ForeColor="#004000" Font-Names="Verdana"
            Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
            BorderColor="#004000" Font-Bold="True" />

<asp:Button runat="server" ForeColor="#004000" Font-Names="Verdana"
            Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
            BorderColor="#004000" Font-Bold="True" BackColor="#FFE0C0" />
```

This is just a sampling of what the `Summer.skin` file should contain. To use it in a real application, you should actually make a definition for each and every server control option. In this case, you have a definition in place for three different types of server controls: Label, TextBox, and Button. After saving the `Summer.skin` file in the `Summer` folder, your file structure should look like Figure 6-5 from the Solution Explorer of Visual Studio 2008.

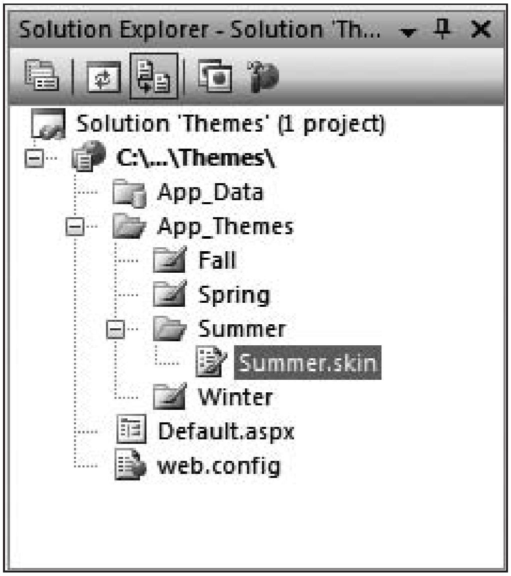


Figure 6-5

As with the regular server control definitions that you put on a typical `.aspx` page, these control definitions must contain the `runat="server"` attribute. If you specify this attribute in the skinned version of the control, you also include it in the server control you put on an `.aspx` page that uses this theme. Also notice that no `ID` attribute is specified in the skinned version of the control. If you specify an `ID` attribute here, you get an error when a page tries to use this theme.

As you can see, you can supply a lot of different visual definitions to these three controls, and this should give the page a *summery* look and feel. An ASP.NET page in this project can then simply use this custom theme as was shown earlier in this chapter (see Listing 6-4).

Listing 6-4: Using the Summer theme in an ASP.NET page

VB

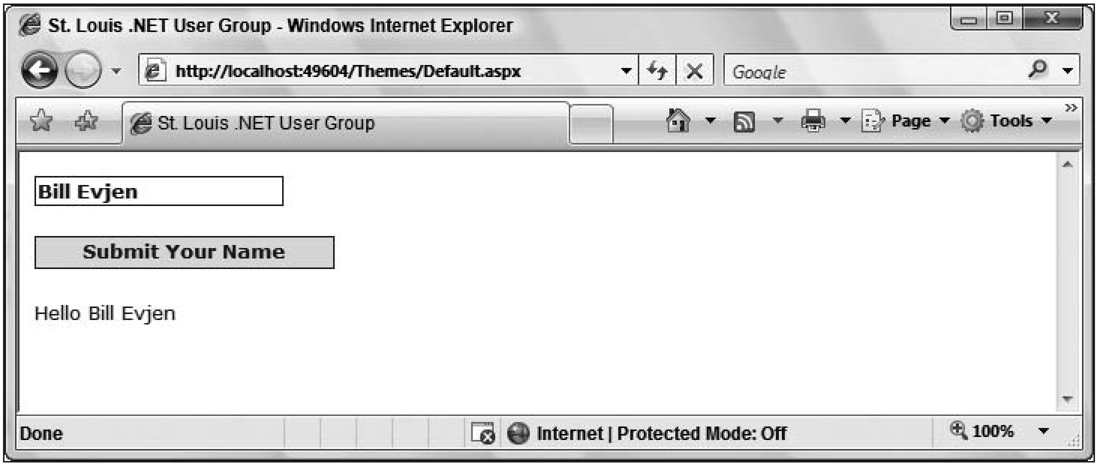
```
<%@ Page Language="VB" Theme="Summer" %>

<script runat="server">
   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
      Label1.Text = "Hello " & TextBox1.Text
   End Sub
</script>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>St. Louis .NET User Group</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Textbox ID="TextBox1" runat="server">
        </asp:Textbox>
        <br />
        <br />
        <asp:Button ID="Button1" runat="server" Text="Submit Your Name"
         OnClick="Button1_Click" />
        <br />
        <br />
        <asp:Label ID="Label1" runat="server" />
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" Theme="Summer" %>

<script runat="server">
    protected void Button1_Click(object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + TextBox1.Text.ToString();
    }
</script>
```

Looking at the server controls on this `.aspx` page, you can see that no styles are associated with them. These are just the default server controls that you drag and drop onto the design surface of Visual Studio 2008. There is, however, the style that you defined in the `Summer.skin` file, as shown in Figure 6-6.



St. Louis .NET User Group - Windows Internet Explorer

http://localhost:49604/Themes/Default.aspx

Bill Evjen

Hello Bill Evjen

Google
St. Louis .NET User Group
Page
Tools
Submit Your Name
Done
Internet | Protected Mode: Off
100%

Figure 6-6

Including CSS Files in Your Themes

In addition to the server control definitions that you create from within a `.skin` file, you can make further definitions using Cascading Style Sheets (CSS). You might have noticed, when using a `.skin` file, that you could define only the styles associated with server controls and nothing else. However, developers usually use quite a bit more than server controls in their ASP.NET pages. For instance, ASP.NET pages are routinely made up of HTML server controls, raw HTML, or even raw text. At present, the *Summer* theme has only a `Summer.skin` file associated with it. Any other items have no style whatsoever applied to them.

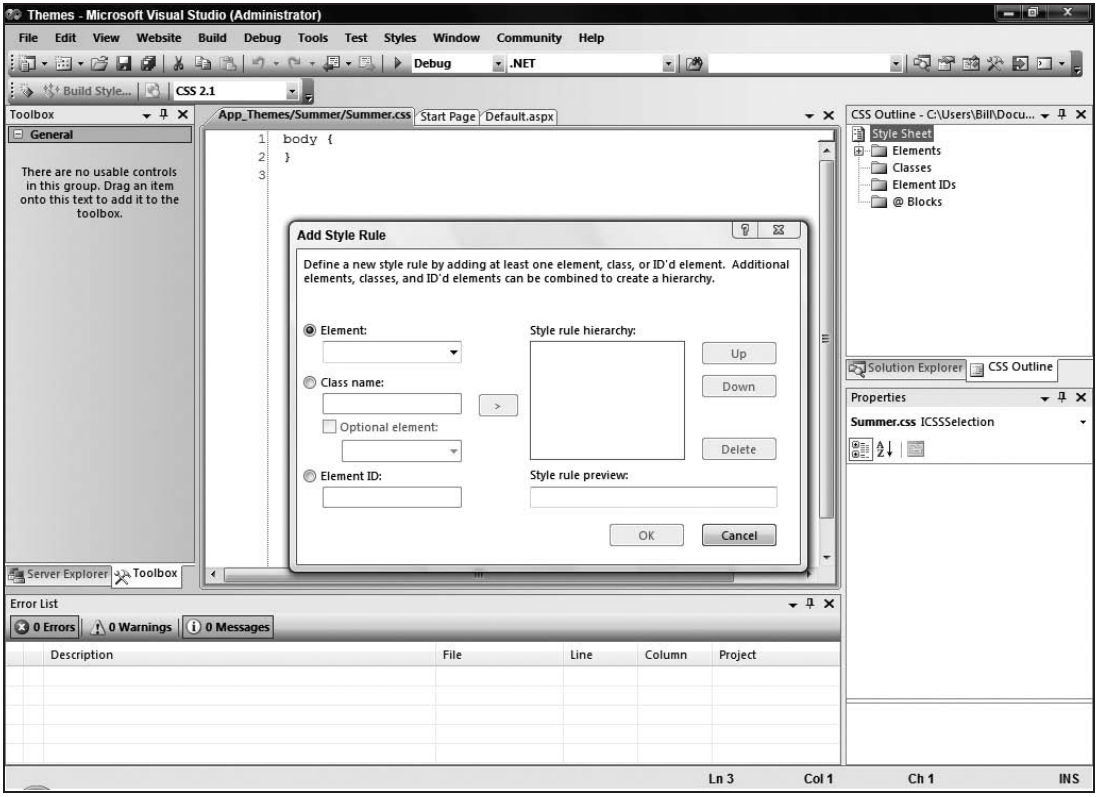
For a theme that goes beyond the server controls, you must further define the theme style so that HTML server controls, HTML, and raw text are all changed according to the theme. You achieve this with a CSS file within your theme folder.

It is rather easy to create CSS files for your themes when using Visual Studio 2008. Right-click the `Summer` theme folder and select Add New Item. In the list of options, select the option Style Sheet and name it `Summer.css`. The `Summer.css` file should be sitting right next to your `Summer.skin` file. This creates an empty `.css` file for your theme. I will not go into the details of how to make a CSS file using Visual Studio 2008 and the CSS creation tool because this was covered earlier in Chapter 2 in this book. The process is also the same as in previous versions of Visual Studio. Just remember that the dialog that comes with Visual Studio 2008 enables you to completely define your CSS page with no need to actually code anything. A sample dialog is shown in Figure 6-7.

To create a comprehensive theme with this dialog, you define each HTML element that might appear in the ASP.NET page or your make use of class names or element IDs. This can be a lot of work, but it is worth it in the end. For now, create a small CSS file that changes some of the non-server control items on your ASP.NET page. This CSS file is shown in Listing 6-5.

Listing 6-5: A CSS file with some definitions

```
body
{
 font-size: x-small;
 font-family: Verdana;
 color: #004000;
}

a:link {
 color: Blue;
 text-decoration: none;
}

a:visited
{
 color: Blue;
 text-decoration: none;
}

a:hover {
 color: Red;
 text-decoration: underline overline;
}
```


Themes - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Tools Test Styles Window Community Help
Debug
.NET
Build Style...
CSS 2.1
Toolbox
General
App_Themes/Summer/Summer.css
Start Page
Default.aspx
CSS Outline - C:\Users\Bill\Docu...
Style Sheet
Elements
Classes
Element IDs
@ Blocks
Add Style Rule
Element:
Class name:
Optional element:
Element ID:
Style rule hierarchy:
Style rule preview:
Up
Down
Delete
OK
Cancel
Solution Explorer
CSS Outline
Properties
Summer.css ICSSSelection
Server Explorer
Toolbox
Error List
0 Errors
0 Warnings
0 Messages
Description
File
Line
Column
Project
Ln 3
Col 1
Ch 1
INS

```
body {
}
```

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

Define a new style rule by adding at least one element, class, or ID'd element. Additional elements, classes, and ID'd elements can be combined to create a hierarchy.

Figure 6-7

In this CSS file, four things are defined. First, you define text that is found within the `<body>` tag of the page (basically all the text). Generally, plenty of text can appear in a typical ASP.NET page that is not placed inside an `<asp:Label>` or `<asp:Literal>` tag. Therefore, you can define how your text should appear in the CSS file; otherwise, your Web page may appear quite odd at times. In this case, a definition is in place for the size, the font family, and the color of the text. You make this definition the same as the one for the `<asp:Label>` server control in the `Summer.skin` file.

The next three definitions in this CSS file revolve around the `<a>` element (for hyperlinks). One cool feature that many Web pages use is responsive hyperlinks — or hyperlinks that change when you hover a mouse over them. The `A:link` definition defines what a typical link looks like on the page. The `A:visited` definition defines the look of the link if the end user has clicked on the link previously (without this definition, it is typically purple in IE). Then the `A:hover` definition defines the appearance of the hyperlink when the end user hovers the mouse over the link. You can see that not only are these three definitions changing the color of the hyperlink, but they are also changing how the underline is used. In fact, when the end user hovers the mouse over a hyperlink on a page using this CSS file, an underline and an overline appear on the link itself.

In CSS files, the order in which the style definitions appear in the `.css` file is important. This is an interpreted file — the first definition in the CSS file is applied first to the page, next the second definition is applied, and so forth. Some styles might change previous styles, so make sure your style definitions are in the proper order. For instance, if you put the `A:hover` style definition first, you would never see it. The `A:link` and `A:visited` definitions would supersede it because they are defined after it. In addition to order, other factors such as the target media type, importance (whether the declaration is specified as important or normal), and the origin of the stylesheet also play a factor in interpreting declarations.

In working with your themes that include `.css` files, you must understand what they can and cannot do for you. For instance, examine an `.aspx` file that contains two text boxes — one text box created using a server control and another text box created using a typical `<input>` HTML element:

```
<asp:Textbox ID="TextBox1" runat="server" />&nbsp;
<input type="text" />
```

Suppose you have a definition for the TextBox server control in the `.skin` file:

```
<asp:Textbox runat="server" ForeColor="#004000" Font-Names="Verdana"
 BackColor="#ffffff" Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
 BorderColor="#004000" Font-Bold="True" />
```

However, what if you also have a definition in your `.css` file for each `<input>` element in the ASP.NET page as shown here:

```
INPUT
{
  background-color: black;
}
```

When you run the `.aspx` page with these kinds of style conflicts, the `.skin` file takes precedence over styles applied to every HTML element that is created using ASP.NET server controls regardless of what the `.css` file says. In fact, this sort of scenario gives you a page in which the `<input>` element that is created from the server control is white, as defined in the `.skin` file, and the second text box is black, as defined in the `.css` file (see Figure 6-8).

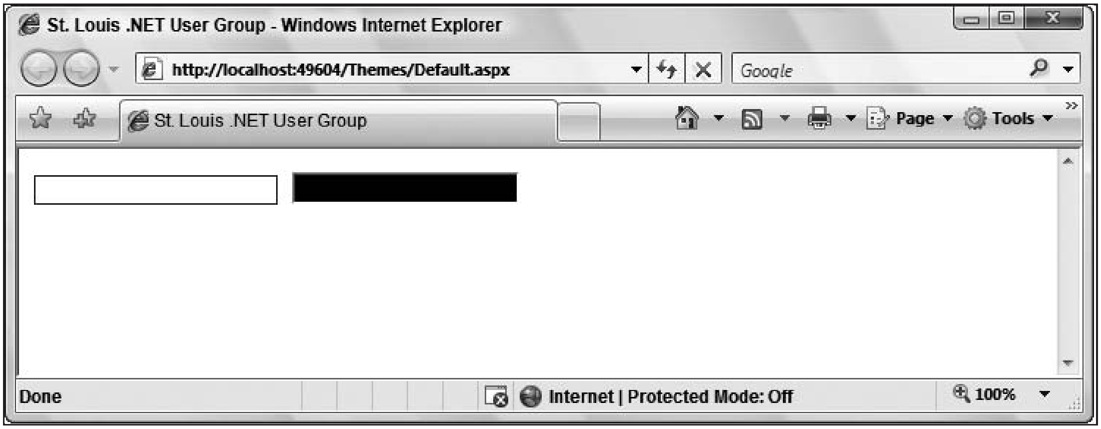

St. Louis .NET User Group - Windows Internet Explorer
http://localhost:49604/Themes/Default.aspx
Google
St. Louis .NET User Group
Page
Tools
Done
Internet | Protected Mode: Off
100%

Figure 6-8

Again, other factors besides the order in which the items are defined can alter the appearance of your page. In addition to order, other factors such as the target media type, importance (whether the declaration is specified as important or normal), and the origin of the stylesheet also play a factor in interpreting declarations.

Having Your Themes Include Images

Probably one of the coolest reasons why themes, rather than CSS, are the better approach for applying a consistent style to your Web page is that themes enable you to incorporate actual images into the style definitions.

Many controls use images to create a better visual appearance. The first step in incorporating images into your server controls that consistently use themes is to create an `Images` folder within the theme folder itself, as illustrated in Figure 6-9.

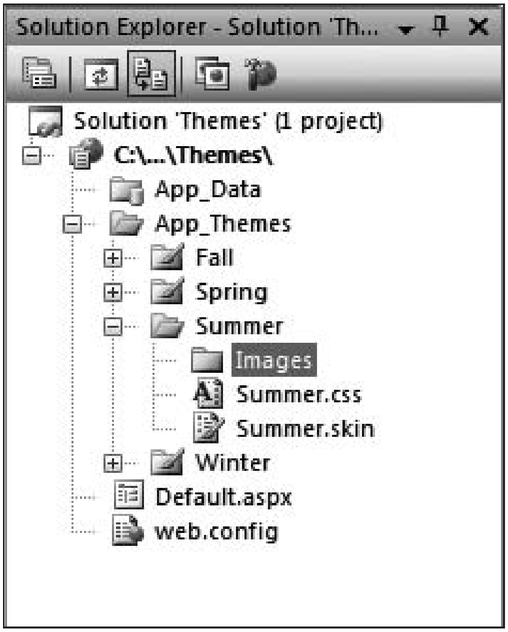

Solution Explorer - Solution 'Th...
Solution 'Themes' (1 project)
C:\...\Themes\
App_Data
App_Themes
Fall
Spring
Summer
Images
Summer.css
Summer.skin
Winter
Default.aspx
web.config

Figure 6-9

You have a couple of easy ways to use the images that you might place in this folder. The first is to incorporate the images directly from the `.skin` file itself. You can do this with the TreeView server control. The TreeView control can contain images used to open and close nodes for navigation purposes. You can place images in your theme for each and every TreeView control in your application. If you do so, you can then define the TreeView server control in the `.skin` file, as shown in Listing 6-6.

Listing 6-6: Using images from the theme folder in a TreeView server control

```
<asp:TreeView runat="server" BorderColor="#FFFFFF" BackColor="#FFFFFF"
   ForeColor="#585880" Font-Size=".9em" Font-Names="Verdana"
   LeafNodeStyle-ImageUrl="images\summer_iconlevel.gif"
   RootNodeStyle-ImageUrl="images\summer_iconmain.gif"
   ParentNodeStyle-ImageUrl="images\summer_iconmain.gif" NodeIndent="30"
   CollapseImageUrl="images\summer_minus.gif"
   ExpandImageUrl="images\summer_plus.gif">
  ...
</asp:TreeView>
```

When you run a page containing a TreeView server control, it is populated with the images held in the `Images` folder of the theme.

It is easy to incorporate images into the TreeView control. The control even specifically asks for an image location as an attribute. The new WebParts controls are used to build portals. Listing 6-7 is an example of a Web Part definition from a `.skin` file that incorporates images from the `Images` folder of the theme.

Listing 6-7: Using images from the theme folder in a WebPartZone server control

```
<asp:WebPartZone ID="WebPartZone1" runat="server"
 DragHighlightColor="#6464FE" BorderStyle="double"
 BorderColor="#E7E5DB" BorderWidth="2pt" BackColor="#F8F8FC"
 cssclass="theme_fadeblue" Font-Size=".9em" Font-Names="Verdana">
    <FooterStyle ForeColor="#585880" BackColor="#CCCCCC"></FooterStyle>
    <HelpVerb ImageURL="images/SmokeAndGlass_help.gif"
     checked="False" enabled="True" visible="True"></HelpVerb>
    <CloseVerb ImageURL="images/SmokeAndGlass_close.gif"
     checked="False" enabled="True" visible="True"></CloseVerb>
    <RestoreVerb ImageURL="images/SmokeAndGlass_restore.gif"
     checked="False" enabled="True" visible="True"></RestoreVerb>
    <MinimizeVerb ImageURL="images/SmokeAndGlass_minimize.gif"
     checked="False" enabled="True" visible="True"></MinimizeVerb>
    <EditVerb ImageURL="images/SmokeAndGlass_edit.gif"
     checked="False" enabled="True" visible="True"></EditVerb>
</asp:WebPartZone>
```

As you can see here, this series of toolbar buttons, which is contained in a WebPartZone control, now uses images that come from the aforementioned `SmokeAndGlass` theme. When this WebPartZone is then generated, the style is defined directly from the `.skin` file, but the images specified in the `.skin` file are retrieved from the `Images` folder in the theme itself.

Not all server controls enable you to work with images directly from the `Themes` folder by giving you an `image` attribute to work with. If you don't have this capability, you must work with the `.skin` file and the CSS file together. If you do, you can place your theme-based images in any element you want. Next is a good example of how to do this.

Place the image that you want to use in the `Images` folder just as you normally would. Then define the use of the images in the `.css` file. The continued `SmokeAndGlass` example in Listing 6-8 demonstrates this.

Listing 6-8: Part of the CSS file from SmokeAndGlass.css

```
.theme_header {
 background-image :url( images/smokeandglass_brownfadetop.gif);
}

.theme_highlighted {
 background-image :url( images/smokeandglass_blueandwhitef.gif);
}

.theme_fadeblue {
 background-image :url( images/smokeandglass_fadeblue.gif);
}
```

These are not styles for a specific HTML element; instead, they are CSS classes that you can put into any HTML element that you want. In this case, each CSS class mentioned here is defining a specific background image to use for the element.

After it is defined in the CSS file, you can utilize this CSS class in the `.skin` file when defining your server controls. Listing 6-9 shows you how.

Listing 6-9: Using the CSS class in one of the server controls defined in the .skin file

```
<asp:Calendar runat="server" BorderStyle="double" BorderColor="#E7E5DB"
 BorderWidth="2" BackColor="#F8F7F4" Font-Size=".9em" Font-Names="Verdana">
    <TodayDayStyle BackColor="#F8F7F4" BorderWidth="1" BorderColor="#585880"
     ForeColor="#585880" />
    <OtherMonthDayStyle BackColor="transparent" ForeColor="#CCCCCC" />
    <SelectedDayStyle ForeColor="#6464FE" BackColor="transparent"
     CssClass="theme_highlighted" />
    <TitleStyle Font-Bold="True" BackColor="#CCCCCC" ForeColor="#585880"
     BorderColor="#CCCCCC" BorderWidth="1pt" CssClass="theme_header" />
    <NextPrevStyle Font-Bold="True" ForeColor="#585880"
     BorderColor="transparent" BackColor="transparent" />
    <DayStyle ForeColor="#000000"
     BorderColor="transparent" BackColor="transparent" />
    <SelectorStyle Font-Bold="True" ForeColor="#696969" BackColor="#F8F7F4" />
    <WeekendDayStyle Font-Bold="False" ForeColor="#000000"
     BackColor="transparent" />
    <DayHeaderStyle Font-Bold="True" ForeColor="#585880"
     BackColor="Transparent" />
</asp:Calendar>
```

This Calendar server control definition from a `.skin` file uses one of the earlier CSS classes in its definition. It actually uses an image that is specified in the CSS file in two different spots within the control (shown in bold). It is first specified in the `<SelectedDayStyle>` element. Here you see the attribute and value `CssClass="theme_highlighted"`. The other spot is within the `<TitleStyle>` element. In this case, it is using `theme_header`. When the control is rendered, these CSS classes are referenced and finally point to the images that are defined in the CSS file.

It is interesting that the images used here for the header of the Calendar control don't really have much to them. For instance, the `smokeandglass_brownfadetop.gif` image that we are using for this example is simply a thin, gray sliver, as shown in Figure 6-10.

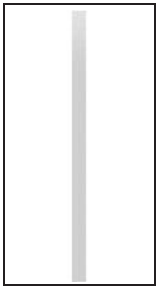
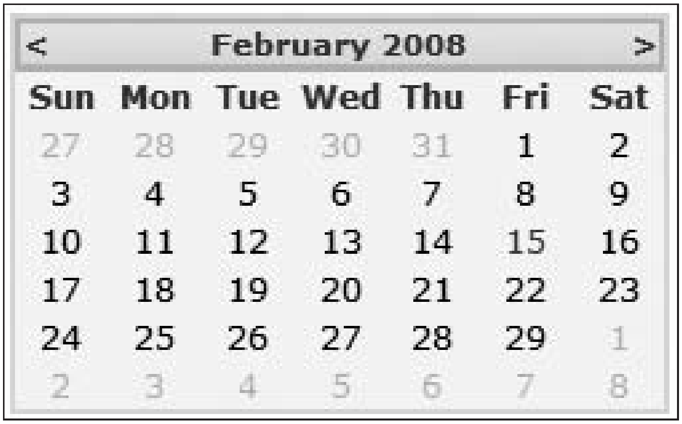


Figure 6-10

This very small image (in this case, very thin) is actually repeated as often as necessary to make it equal the length of the header in the Calendar control. The image is lighter at the top and darkens toward the bottom. Repeated horizontally, this gives a three-dimensional effect to the control. Try it out, and you can get the result shown in Figure 6-11.



< February 2008 >

Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	1
2	3	4	5	6	7	8

Figure 6-11

Defining Multiple Skin Options

Using the themes technology in ASP.NET, you can have a single theme; but also, within the theme's `.skin` file, you can have specific controls that are defined in multiple ways. You can frequently take advantage of this feature within your themes. For instance, you might have text box elements scattered throughout your application, but you might not want each and every text box to have the same visual appearance. In this case, you can create multiple versions of the `<asp:Textbox>` server control within your `.skin` file. In Listing 6-10 you see how to create multiple versions of the `<asp:Textbox>` control in the `.skin` file from Listing 6-3.

Listing 6-10: The Summer.skin file, which contains multiple versions of the <asp:Textbox> server control

```
<asp:Label runat="server" ForeColor="#004000" Font-Names="Verdana"
            Font-Size="X-Small" />

<asp:Textbox runat="server" ForeColor="#004000" Font-Names="Verdana"
            Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
            BorderColor="#004000" Font-Bold="True" />

<asp:Textbox runat="server" ForeColor="#000000" Font-Names="Verdana"
            Font-Size="X-Small" BorderStyle="Dotted" BorderWidth="5px"
            BorderColor="#000000" Font-Bold="False" SkinID="TextboxDotted" />

<asp:Textbox runat="server" ForeColor="#000000" Font-Names="Arial"
            Font-Size="X-Large" BorderStyle="Dashed" BorderWidth="3px"
            BorderColor="#000000" Font-Bold="False" SkinID="TextboxDashed" />

<asp:Button runat="server" ForeColor="#004000" Font-Names="Verdana"
            Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
            BorderColor="#004000" Font-Bold="True" BackColor="#FFE0C0" />
```

In this `.skin` file, you can see three definitions in place for the TextBox server control. The first one is the same as before. Although the second and third definitions have a different style, they also contain a new attribute in the definition — `SkinID`. To create multiple definitions of a single element, you use the `SkinID` attribute to differentiate among the definitions. The value used in the `SkinID` can be anything you want. In this case, it is `TextboxDotted` and `TextboxDashed`.

Note that no `SkinID` attribute is used for the first `<asp:Textbox>` definition. By not using one, you are saying that this is the default style definition to use for each `<asp:Textbox>` control on an ASP.NET page that uses this theme but has no pointer to a particular `SkinID`.

Take a look at a sample `.aspx` page that uses this `.skin` file in Listing 6-11.

Listing 6-11: A simple .aspx page that uses the Summer.skin file with multiple text-box style definitions

```
<%@ Page Language="VB" Theme="Summer" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Different SkinIDs</title>
</head>
<body>
    <form id="form1" runat="server">
    <p>
        <asp:Textbox ID="TextBox1" runat="server">Textbox1</asp:Textbox>
    </p><p>
        <asp:Textbox ID="TextBox2" runat="server"
         SkinId="TextboxDotted">Textbox2</asp:Textbox>
    </p><p>
```

Continued

```
            <asp:Textbox ID="TextBox3" runat="server"
             SkinId="TextboxDashed">Textbox3</asp:Textbox>
        </p>
        </form>
    </body>
    </html>
```

This small `.aspx` page shows three text boxes, each of a different style. When you run this page, you get the results shown in Figure 6-12.

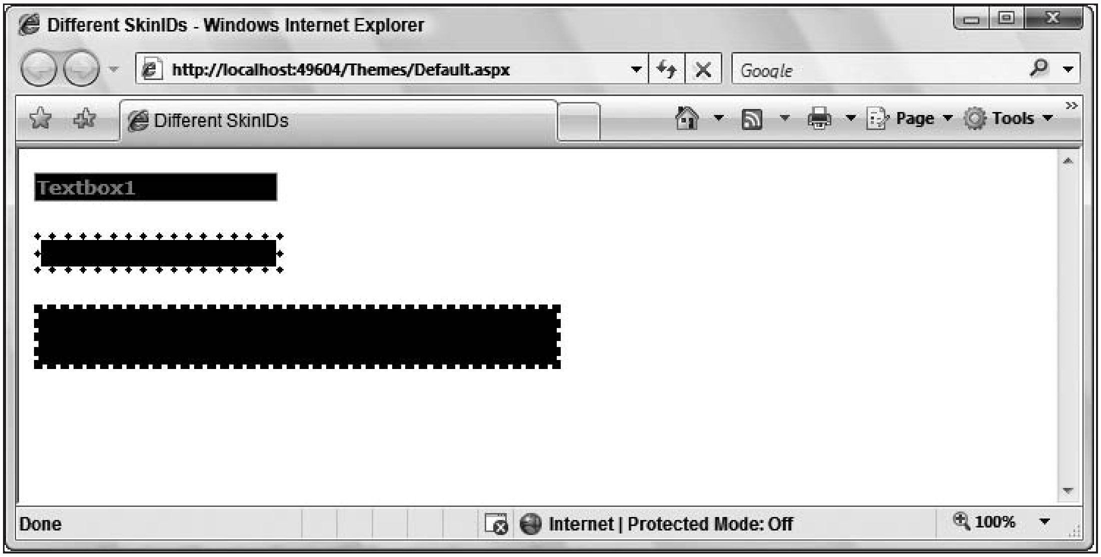

Different SkinIDs - Windows Internet Explorer
http://localhost:49604/Themes/Default.aspx
Google
Different SkinIDs
Page
Tools
Textbox1
Done
Internet | Protected Mode: Off
100%

Figure 6-12

The first text box does not point to any particular `SkinID` in the `.skin` file. Therefore, the default skin is used. As stated before, the default skin is the one in the `.skin` file that does not have a `SkinID` attribute in it. The second text box then contains `SkinID="TextboxDotted"` and, therefore, inherits the style definition defined in the `TextboxDotted` skin in the `Summer.skin` file. The third text box takes the `SkinID` `TextboxDashed` and is also changed appropriately.

As you can see, it is quite simple to define multiple versions of a control that can be used throughout your entire application.

Programmatically Working with Themes

So far, you have seen examples of working with ASP.NET themes in a declarative fashion, but you can also work with themes programmatically.

Assigning the Page's Theme Programmatically

To programmatically assign the theme to the page, use the construct shown in Listing 6-12.

Listing 6-12: Assigning the theme of the page programmatically

VB

```
<script runat="server">
    Protected Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs)
       Page.Theme = Request.QueryString("ThemeChange")
    End Sub
</script>
```

C#

```
<script runat="server">
    protected void Page_PreInit(object sender, System.EventArgs e)
    {
       Page.Theme = Request.QueryString["ThemeChange"];
    }
</script>
```

You must set the `Theme` of the `Page` property in or before the `Page_PreInit` event for any static controls that are on the page. If you are working with dynamic controls, set the `Theme` property before adding it to the Controls collection.

Assigning a Control's SkinID Programmatically

Another option is to assign a specific server control's `SkinID` property programmatically (see Listing 6-13).

Listing 6-13: Assigning the server control's SkinID property programmatically

VB

```
<script runat="server">
    Protected Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs)
       TextBox1.SkinID = "TextboxDashed"
    End Sub
</script>
```

C#

```
<script runat="server">
    protected void Page_PreInit(object sender, System.EventArgs e)
    {
       TextBox1.SkinID = "TextboxDashed";
    }
</script>
```

Again, you assign this property before or in the `Page_PreInit` event in your code.

Themes, Skins, and Custom Controls

If you are building custom controls in an ASP.NET world, understand that end users can also apply themes to the controls that they use in their pages. By default, your custom controls are theme-enabled whether your custom control inherits from `Control` or `WebControl`.

To disable theming for your control, you can simply use the `Themeable` attribute on your class. This is illustrated in Listing 6-14.

Listing 6-14: Disabling theming for your custom controls

VB

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Text
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls

<DefaultProperty("HeaderText"), _
 ToolboxData("<{0}:WebCustomControl1 runat=server></{0}:WebCustomControl1>"), _
  Themeable(False)> _
 Public Class WebCustomControl1
     Inherits WebControl

     <Bindable(True), Category("Appearance"), DefaultValue("Enter Value"), _
      Localizable(True)> Property HeaderText() As String
         Get
             Dim s As String = CStr(ViewState("HeaderText"))
             If s Is Nothing Then
                 Return String.Empty
             Else
                 Return s
             End If
         End Get

         Set(ByVal Value As String)
             ViewState("HeaderText") = Value
         End Set
     End Property


     Protected Overrides Sub RenderContents(ByVal output As HtmlTextWriter)
         output.Write("<h1>" & HeaderText & "<h1>")
     End Sub

 End Class
```

C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ControlForThemes
{
        [DefaultProperty("HeaderText")]
        [ToolboxData("<{0}:WebCustomControl1 runat=server></{0}:WebCustomControl1>")]
        [Themeable(false)]
        public class WebCustomControl1 : WebControl
        {
            [Bindable(true)]
            [Category("Appearance")]
            [DefaultValue("Enter Value")]
            [Localizable(true)]
            public string HeaderText
            {
                get
                {
                    String s = (String)ViewState["HeaderText"];
                    return ((s == null) ? String.Empty : s);
                }

                set
                {
                    ViewState["HeaderText"] = value;
                }
            }

            protected override void RenderContents(HtmlTextWriter output)
            {
                output.Write("<h1>" + HeaderText + "<h1>");
            }
        }
    }
```

Looking over the code from the above example, you can see that theming was disabled by applying the `Themeable` attribute to the class and setting it to False.

You can use a similar approach to disable theming for the individual properties that might be in your custom controls. You do this as illustrated in Listing 6-15.

Listing 6-15: Disabling theming for properties in your custom controls

VB

```
    <Bindable(True), Category("Appearance"), DefaultValue("Enter Value"), _
     Localizable(True), Themeable(False)> Property HeaderText() As String
        Get
            Dim s As String = CStr(ViewState("HeaderText"))
            If s Is Nothing Then
                Return String.Empty
            Else
                Return s
            End If
        End Get

        Set(ByVal Value As String)
            ViewState("HeaderText") = Value
```

Continued

```
        End Set
    End Property
```

C#

```
        [Bindable(true)]
        [Category("Appearance")]
        [DefaultValue("Enter Value")]
        [Themeable(false)]
        public string HeaderText
        {
            get
            {
                String s = (String)ViewState["HeaderText"];
                return ((s == null) ? String.Empty : s);
            }

            set
            {
                ViewState["HeaderText"] = value;
            }
        }
```

In this case, you set the `Themeable` attribute at the property level to `False` in the same manner as you did at the class level.

If you have enabled themes for these items, how would you go about applying a theme definition to a custom control? For this example, use the custom server control shown in Listing 6-14, but set the `Themeable` attributes to `True`. Next, create a `.skin` file in a theme and add the control to the theme as you would any other ASP.NET server control. This is illustrated in Listing 6-16.

Listing 6-16: Changing properties in a custom control in the .skin file

```
<%@ Register Assembly="ControlForThemes" Namespace="ControlForThemes"
    TagPrefix="cc1" %>
<cc1:webcustomcontrol1 runat="server" HeaderText="FROM THE SKIN FILE" />
```

When defining custom server controls in your themes, you use the same approach as you would when placing a custom server control inside of a standard ASP.NET `.aspx` page. In Listing 6-16, you can see that the custom server control is registered in the `.skin` file using the `@Register` page directive. This directive gives the custom control a `TagPrefix` value of `cc1`. Note that the `TagPrefix` values presented in this page can be different from those presented in any other `.aspx` page that uses the same custom control. The only things that have to be the same are the `Assembly` and `Namespace` attributes that point to the specific control being defined in the file. Also note the control definition in the skin file, as with other standard controls, does not require that you specify an `ID` attribute, but only the `runat` attribute along with any other property that you wish to override.

Next, create a standard `.aspx` page that uses your custom server control. Before running the page, be sure to apply the defined theme on the page using the `Theme` attribute in the `@Page` directive. With everything in place, running the page produces the following results in the browser:

```
FROM THE SKIN FILE
```

This value, which was specified in the skin file, is displayed no matter what you apply as the `HeaderText` value in the server control.

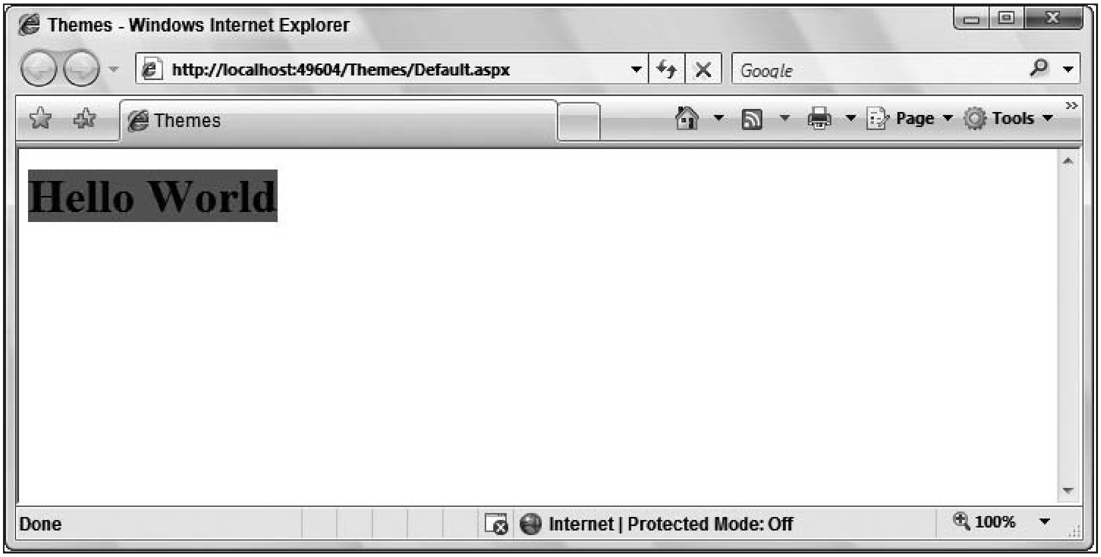
In addition to changing values of custom properties that are contained in server control, you can also change the inherited properties that come from `WebControl`. For instance, you can change settings in your skin file as shown in Listing 6-17.

Listing 6-17: Changing inherited properties in the custom control

```
<%@ Register Assembly="ControlForThemes" Namespace="ControlForThemes"
    TagPrefix="cc1" %>
<cc1:webcustomcontrol1 runat="server" BackColor="Gray" />
```

With this in place, you have changed one of the inherited properties from the skin file. This setting changes the background color of the server control to gray (even if it is set to something else in the control itself). The result is presented in Figure 6-13.

You can find more information on building your own custom server controls in Chapter 26.


Themes - Windows Internet Explorer
http://localhost:49604/Themes/Default.aspx
Google
Themes
Page
Tools
Done
Internet | Protected Mode: Off
100%

Hello World

Figure 6-13

Summary

With the availability of themes and skins in ASP.NET 3.5, it is quite easy to apply a consistent look and feel across your entire application. Remember that themes can contain only simple server control definitions in a `.skin` file or elaborate style definitions, which include not only `.skin` files, but also CSS style definitions and even images!

As you will see later in the book, you can use themes in conjunction with the personalization features that ASP.NET provides. This enables your end users to customize their experiences by selecting their own themes. Your application can present a theme just for them, and it can remember their choices through the APIs that are offered in ASP.NET 3.5.

7

Data Binding in ASP.NET 3.5

One of the most exciting features of ASP.NET 1.0/1.1 was its capability to bind entire collections of data to controls at runtime without requiring you to write large amounts of code. The controls understood they were data-bound and would render the appropriate HTML for each item in the data collection. Additionally, you could bind the controls to any type of data sources, from simple arrays to complex Oracle database query results. This was a huge step forward from ASP, in which each developer was responsible for writing all the data access code, looping through a recordset, and manually rendering the appropriate HTML code for each record of data.

In ASP.NET 2.0, Microsoft took the concept of data binding and expanded it to make it even easier to understand and use by introducing a new layer of data abstraction called data source controls. It also brought into the toolbox a series of new and powerful databound controls such as the GridView, DetailsView, and FormView. ASP.NET 3.5 continues to make fetching and displaying data in ASP.NET as simple as possible by introducing the LinqDataSource control and the ListView control

This chapter explores all the provided data source controls, and describes other data-binding features in ASP.NET. It shows how you can use the data source controls to easily and quickly bind data to data-bound controls. It also focuses on the power of the data-bound list controls included in ASP.NET 3.5, such as the GridView, DetailsView, FormView, and the new ListView control. Finally, you take a look at changes in the inline data binding syntax and inline XML data binding.

Data Source Controls

In ASP.NET 1.0/1.1, you typically performed a data-binding operation by writing some data access code to retrieve a `DataReader` or a `DataSet` object; then you bound that data object to a server control such as a DataGrid, DropDownList, or ListBox. If you wanted to update or delete the bound data, you were then responsible for writing the data access code to do that. Listing 7-1 shows a typical example of a data-binding operation in ASP.NET 1.0/1.1.

Listing 7-1: Typical data-binding operation in ASP.NET 1.0/1.1

VB

```
Dim conn As New SqlConnection()
Dim cmd As New SqlCommand("SELECT * FROM Customers", conn)

Dim da As New SqlDataAdapter(cmd)

Dim ds As New DataSet()
da.Fill(ds)

DataGrid1.DataSource = ds
DataGrid1.DataBind()
```

C#

```
SqlConnection conn = new SqlConnection();
SqlCommand cmd = new SqlCommand("SELECT * FROM Customers", conn);

SqlDataAdapter da = new SqlDataAdapter(cmd);

DataSet ds = new DataSet();
da.Fill(ds);

DataGrid1.DataSource = ds;
DataGrid1.DataBind();
```

ASP.NET 2.0 introduced an additional layer of abstraction through the use of data source controls. As shown in Figure 7-1, these controls abstract the use of an underlying data provider, such as the SQL Data Provider or the OLE DB Data Provider. This means you no longer need to concern yourself with the hows and whys of using the data providers. Instead, the data source controls do all the heavy lifting for you. You need to know only where your data is and, if necessary, how to construct a query for performing CRUD (Create, Retrieve, Update, and Delete) operations.

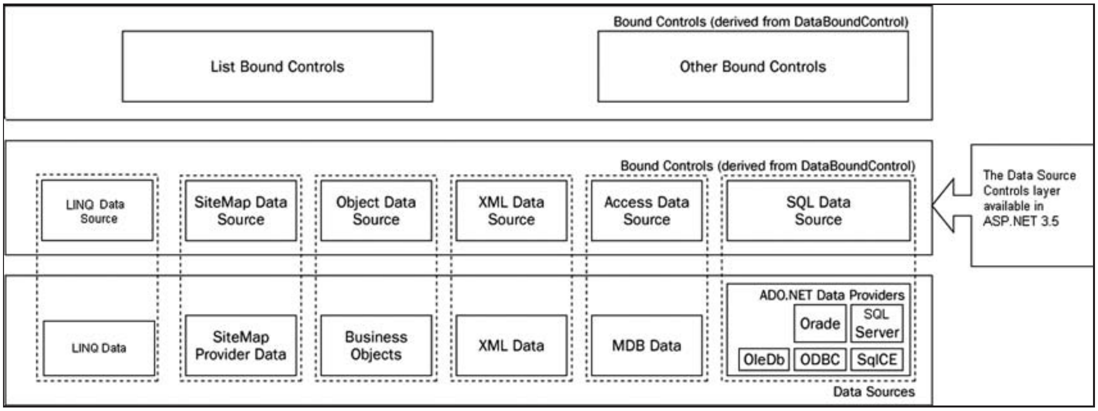

Bound Controls (derived from DataBoundControl)
List Bound Controls
Other Bound Controls
Bound Controls (derived from DataBoundControl)
LINQ Data Source
SiteMap Data Source
Object Data Source
XML Data Source
Access Data Source
SQL Data Source
The Data Source Controls layer available in ASP.NET 3.5
LINQ Data
SiteMap Provider Data
Business Objects
XML Data
MDB Data
ADO.NET Data Providers
Oracle
SQL Server
OleDb
ODBC
SqlCE
Data Sources

Figure 7-1

Additionally, because the data source controls all derive from the `Control` class, you can use them much as you would any other Web Server control. For instance, you can define and control the behavior of the data source control either declaratively in your HTML or programmatically. This means you can perform all manner of data access and manipulation without ever having to write one line of code. In fact, although you certainly can control the data source controls from code, the samples in this chapter show you how to perform powerful database queries using nothing more than the Visual Studio 2008 wizards and declarative syntax.

The six built-in data source controls in ASP.NET 3.5 are each used for a specific type of data access. The following table lists and describes each data source control.

Control Name	**Description**
SqlDataSource control	Provides access to any data source that has an ADO.NET Data Provider available; by default, the control has access to the ODBC, OLE DB, SQL Server, Oracle, and SQL Server CE providers.
LinqDataSource control	Provides access to different types of data objects using LINQ queries.
ObjectDataSource control	Provides specialized data access to business objects or other classes that return data.
XmlDataSource control	Provides specialized data access to XML documents, either physically or in-memory.
SiteMapDataSource control	Provides specialized access to site map data for a Web site that is stored by the site map provider.
AccessDataSource control	Provides specialized access to Access databases.

All the data source controls are derived from the `DataSourceControl` class, which is derived from `Control` and implements the `IDataSource` and `IListSource` interfaces. This means that although each control is designed for use with specific data sources, all data source controls share a basic set of core functionality. It also means that it is easy for you to create your own custom data source controls based on the structure of your specific data sources.

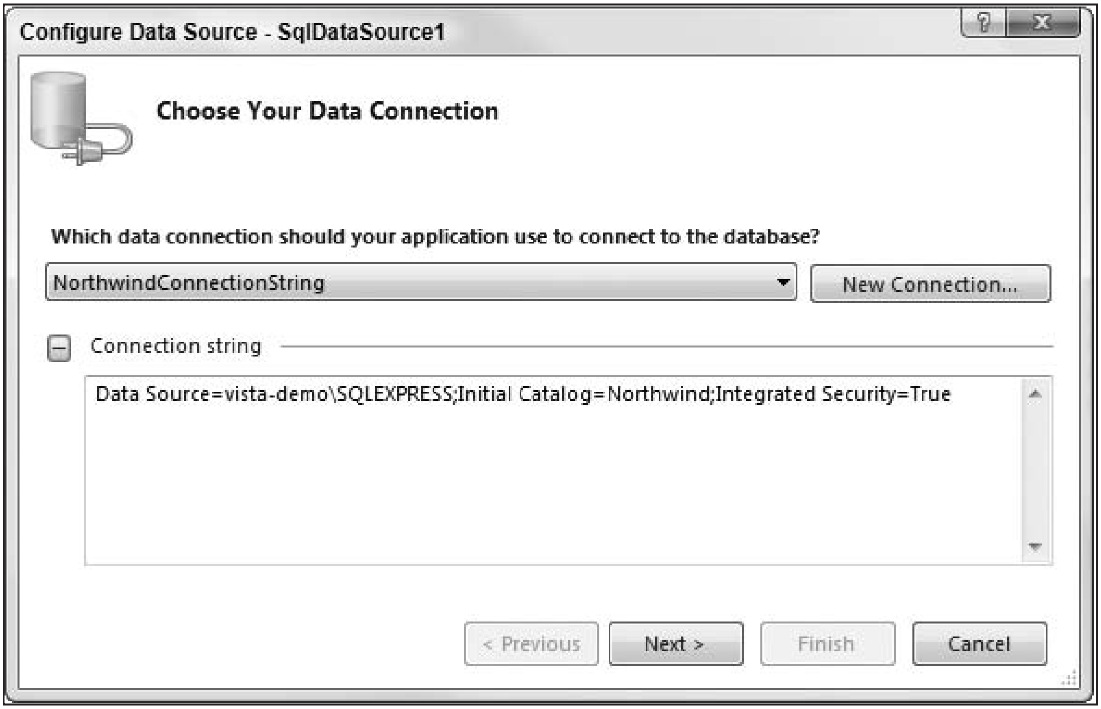
SqlDataSource Control

The SqlDataSource control is the data source control to use if your data is stored in a SQL Server, SQL Server Express, Oracle Server, ODBC data source, OLE DB data source, or Windows SQL CE Database. The control provides an easy-to-use wizard that walks you through the configuration process, or you can modify the control manually by changing the control attributes directly in Source view. In the example presented in this section, you walk through creating a SqlDataSource control and configuring it using the wizard. After you complete the configuration, you examine the source code it generates.

Begin using the control by opening an `.aspx` page inside a Visual Studio Web site project and dragging the SqlDataSource control from the toolbox onto the form. The Visual Studio toolbox has been divided into functional groups so you find all the data-related controls located under the Data section.

Configuring a Data Connection

After the control has been dropped onto the Web page, you tell it what connection it should use. The easiest way to do this is by using the Configure Data Source Wizard, shown in Figure 7-2. Launch this wizard by selecting the Configure Data Source option from the data source control's smart tag menu.



Configure Data Source - SqlDataSource1

Choose Your Data Connection

Which data connection should your application use to connect to the database?

NorthwindConnectionString

Connection string

Data Source=vista-demo\SQLEXPRESS;Initial Catalog=Northwind;Integrated Security=True

Figure 7-2

Once the wizard opens, you should create a connection to the Northwind database in SQL. You will use this connection for most of the demonstrations in this chapter.

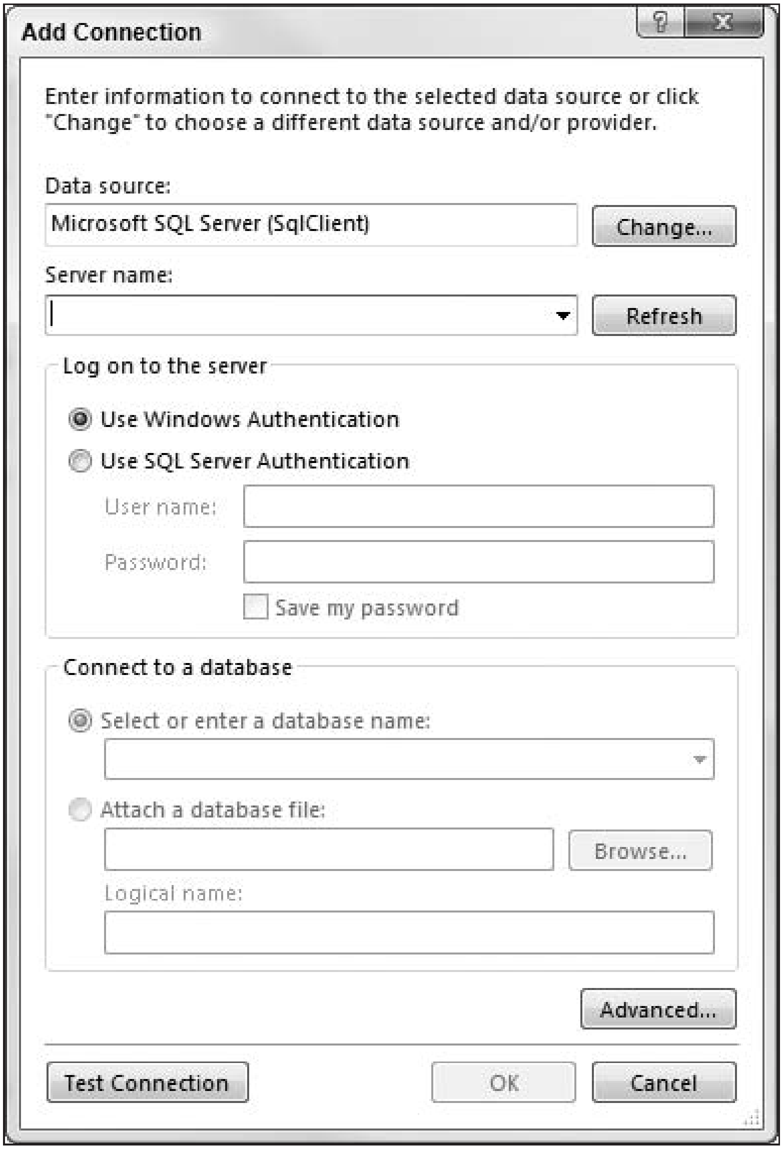
> *Beginning with Microsoft SQL Server 2005, Microsoft no longer includes the Northwind sample database as part of the standard installation. You can still download the installation scripts for the sample database from the following location:*
>
> `http://www.microsoft.com/downloads/details.aspx?FamilyId=06616212-0356-46A0-8DA2-EEBC53A68034&displaylang=en`

After the wizard opens, you can select an existing connection from the drop-down list or create a new connection. If you click the New Connection button, the Connection Properties dialog, shown in Figure 7-3, appears. From here, you can set all the properties of a new database connection.

Click the Change button. From here, you can choose the specific data provider you want this connection to use. By default, the control uses the ADO.NET SQL Data Provider; also available are Oracle, OLE DB, ODBC, and SQL Server Mobile Edition providers.

> *The list of providers is generated from the data contained in the* `DbProviderFactory` *node of the* `machine.config` *file. If you have additional providers to display in the wizard you can modify your* `machine.config` *file to include specific providers' information.*

Next, simply fill in the appropriate information for your database connection. Click the Test Connection button to verify that your connection information is correct, and then click OK to return to the wizard.



Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient)

Server name:

Log on to the server

● Use Windows Authentication

○ Use SQL Server Authentication

User name:

Password:

☐ Save my password

Connect to a database

● Select or enter a database name:

○ Attach a database file:

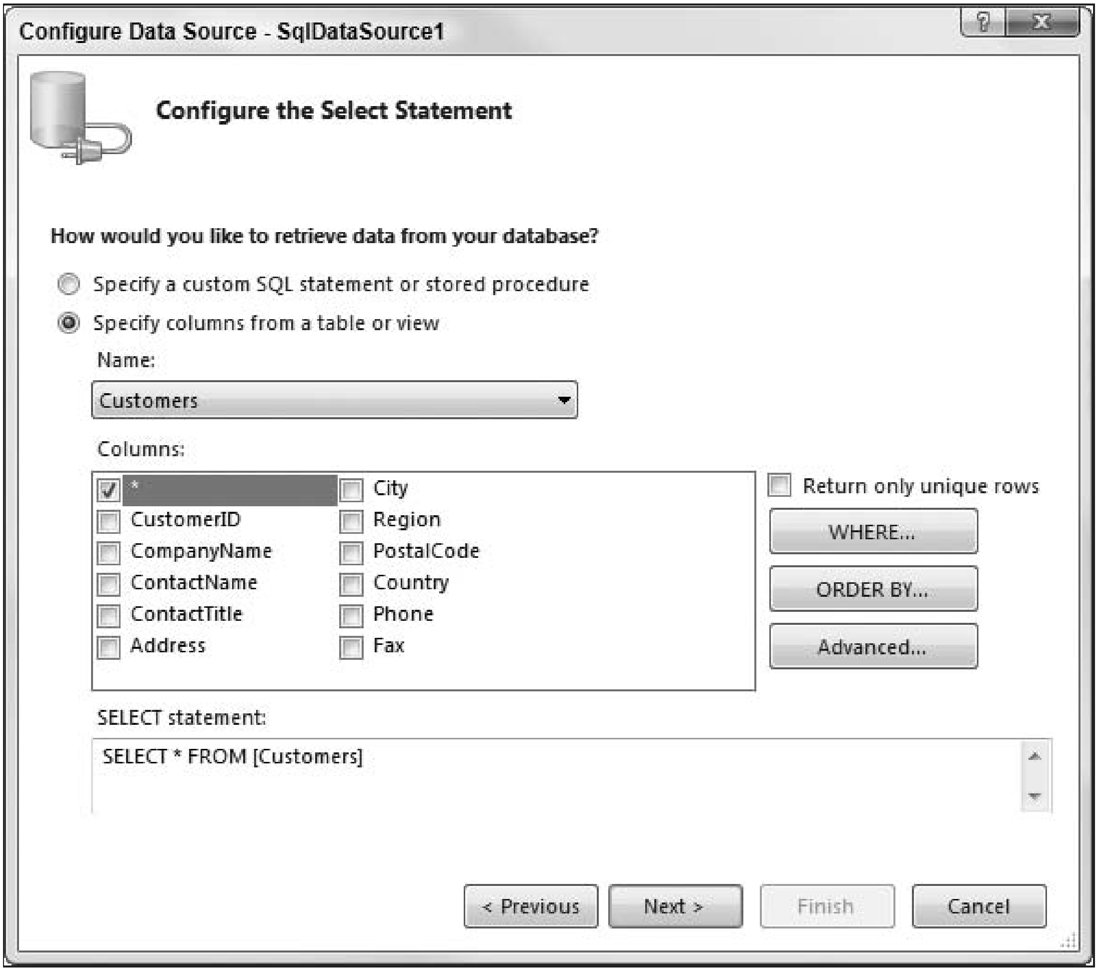
Logical name:

Figure 7-3

After you have returned to the Data Source Configuration Wizard, notice that the connection you created is listed in the available connections drop-down list. After you select a connection string from the drop-down, the connection information shows in the Data Connection info area. This allows you to easily review the connection information for the Connection selected in the drop-down list.

Click the Next button to continue through the wizard. The next step allows you to choose to have the wizard save your connection information in your `web.config` file to make maintenance and deployment of your application easier. This screen allows you to specify the key under which the connection information should be stored in the configuration file. Should you choose not to store your connection information in the `web.config` file, it is stored in the actual `.aspx` page as a property of the `SqlDataSource` control named `ConnectionString`. If the provider chosen was not the SQL Data Provider, a property named `ProviderName` will be used to store that setting.

The next step in the wizard allows you to configure the `SELECT` statement your data source control will use to retrieve data from the database. This screen, shown in Figure 7-4, gives you a drop-down list of all the tables and views available in the database that you specified in your connection information. After you select a table or view, the list box allows you to select the column you want to include in the query. You can select all columns available using an asterisk (`*`), or you can choose specific columns by marking the check box located next to each column name. By clicking the WHERE or ORDER BY button, it is also possible to specify `WHERE` clause parameters for filtering and `ORDER BY` settings for sorting in your query. For now, do not enter any additional `WHERE` or `ORDER BY` settings.



Configure Data Source - SqlDataSource1

Configure the Select Statement

How would you like to retrieve data from your database?

○ Specify a custom SQL statement or stored procedure

● Specify columns from a table or view

Name: Customers

Columns:

- ☑ *
- ☐ CustomerID
- ☐ CompanyName
- ☐ ContactName
- ☐ ContactTitle
- ☐ Address
- ☐ City
- ☐ Region
- ☐ PostalCode
- ☐ Country
- ☐ Phone
- ☐ Fax

☐ Return only unique rows

SELECT statement:

```
SELECT * FROM [Customers]
```

Figure 7-4

Finally, the Advanced button contains two advanced options. You can have the wizard generate `INSERT`, `UPDATE`, and `DELETE` statements for your data, based on the `SELECT` statement you created. You can also configure the data source control to use Optimistic Concurrency to prevent data concurrency issues.

Optimistic Concurrency is a database technique that can help you prevent the accidental overwriting of data. When Optimistic Concurrency is enabled, the Update and Delete SQL statements used by the SqlDataSource control are modified so that they include both the original and updated values. When the queries are executed, the data in the targeted record is compated to the SqlDataSource controls original values and if a difference is found,indicating that the data has changed since it was originally reterived by the SqlDataSource control, the Update or Delete will not occur.

The final screen of the wizard allows you to preview the data selected by your data source control to verify the query is working as you expect it to. Simply click the Finish button to complete the wizard.

When you are done configuring your data connection, you can see exactly what the configured SqlDataSource control looks like. Change to Source view in Visual Studio to see how the wizard has generated the appropriate attributes for your control. It should look something like the code in Listing 7-2.

Listing 7-2: Typical SqlDataSource control generated by Visual Studio

```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server"
    SelectCommand="SELECT * FROM [Customers]"
    ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>">
</asp:SqlDataSource>
```

You can see that the control uses a declarative syntax to configure which connection it should use by creating a `ConnectionString` attribute, and what query to execute by creating a `SelectCommand` attribute. A little later in this chapter, you look at how to configure the `SqlDataSource` control to execute `INSERT`, `UPDATE`, and `DELETE` commands as this data changes.

Data Source Mode Property

One of many important properties of the SqlDataSource control is the `DataSourceMode` property. This property enables you to tell the control if it should use a `DataSet` or a `DataReader` internally when retrieving the data. This is important when you are designing data-driven ASP.NET pages. If you choose to use a `DataReader`, data is retrieved using what is commonly known as *fire hose mode*, or a forward-only, read-only cursor. This is the fastest and most efficient way to read data from your data source because a `DataReader` does not have the memory and processing overhead of a `DataSet`. But choosing to use a `DataSet` makes the data source control more powerful by enabling the control to perform other operations such as filtering, sorting, and paging. It also enables the built-in caching capabilities of the control. Each option offers distinct advantages and disadvantages, so consider this property carefully when designing your Web site. The default value for this property is to use a `DataSet` to retrieve data. The code in Listing 7-3 shows how to add the `DataSourceMode` property to your SqlDataSource control.

Listing 7-3: Adding the DataSourceMode property to a SqlDataSource control

```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server"
    SelectCommand="SELECT * FROM [Customers]"
    ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>"
     DataSourceMode="DataSet">
</asp:SqlDataSource>
```

Filtering Data Using SelectParameters

Of course, when selecting data from your data source, you may not want to get every single row of data from a view or table. You want to be able to specify parameters in your query to limit the data that is returned. The data source control allows you to do this by using the `SelectParameters` collection to create parameters that it can use at runtime to alter the data that is returned from a query.

The `SelectParameters` collection consists of types that derive from the `Parameters` class. You can combine any number of parameters in the collection. The data source control then uses these to create a dynamic SQL query. The following table lists and describes the available parameter types.

Parameter	Description
`ControlParameter`	Uses the value of a property of the specified control
`CookieParameter`	Uses the key value of a cookie
`FormParameter`	Uses the key value from the Forms collection
`QuerystringParameter`	Uses a key value from the QueryString collection
`ProfileParameter`	Uses a key value from the user's profile
`SessionParameter`	Uses a key value from the current user's session

Because all the parameter controls derive from the `Parameter` class, they all contain several useful common properties. Some of these properties are shown in the following table.

Property	Description
`Type`	Allows you to strongly type the value of the parameter
`ConvertEmptyStringToNull`	Indicates the control should convert the value assigned to it to `Null` if it is equal to `System.String.Empty`
`DefaultValue`	Allows you to specify a default value for the parameter if it is evaluated as `Null`

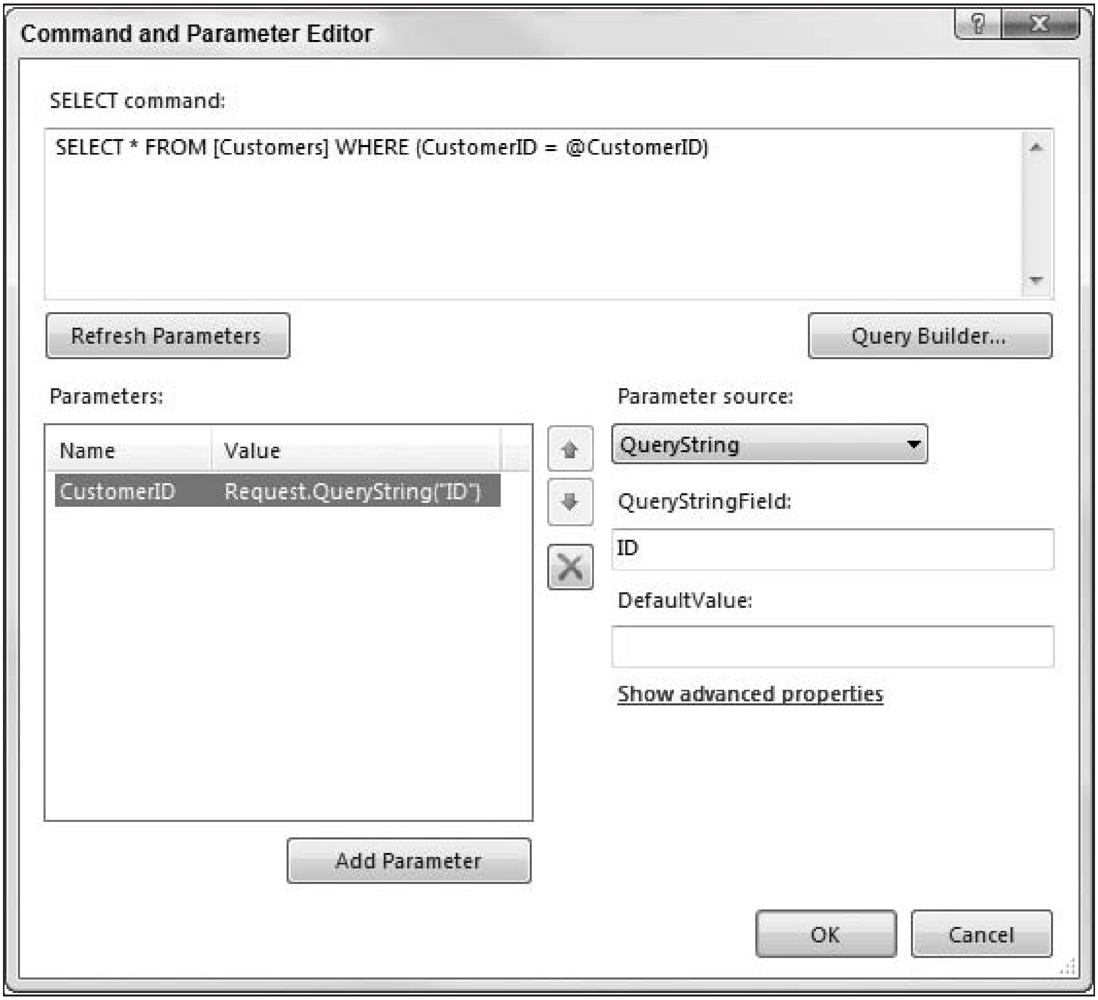
The code in Listing 7-4 shows an example of adding a QueryStringParameter to the `SelectParameters` collection of your SqlDataSource control. As you can see, the `SelectCommand` query has been modified to include a `WHERE` clause. When you run this code, the value of the query string field `ID` is bound to the `@CustomerID` placeholder in your `SelectCommand`, allowing you to select only those customers whose `CustomerID` field matches the value of the query string field.

Listing 7-4: Filtering select data using SelectParameter controls

```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server"
    SelectCommand="SELECT * FROM [Customers] WHERE ([CustomerID] = @CustomerID)"
    ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>"
    DataSourceMode="DataSet">
```

```
        <SelectParameters>
            <asp:QueryStringParameter Name="CustomerID"
                QueryStringField="ID" Type="String">
            </asp:QueryStringParameter>
        </SelectParameters>
   </asp:SqlDataSource>
```

In addition to hand-coding your `SelectParameters` collection, you can create parameters using the Command and Parameter Editor dialog, which can be accessed by modifying the `SelectQuery` property of the SqlDataSource control while you are viewing the Web page in design mode. Figure 7-5 shows the Command and Parameter Editor dialog.


Command and Parameter Editor
Refresh Parameters
Query Builder...
Add Parameter
Show advanced properties
OK
Cancel

SELECT command:

```
SELECT * FROM [Customers] WHERE (CustomerID = @CustomerID)
```

Parameters:

Name	Value
CustomerID	Request.QueryString("ID")

Parameter source: QueryString

QueryStringField: ID

DefaultValue:

Figure 7-5

This dialog gives you a fast and friendly way to create `SelectParameters` for your query. Simply select the Parameter source from the drop-down list and enter the required parameter data. Figure 7-5 demonstrates how to add the `QuerystringParameter` (based on the value of the `querystring` Field ID) to your SqlDataSource control.

Conflict Detection Property

The `ConflictDetection` property allows you to tell the SqlDataSource control what style of conflict detection to use when updating the data. This determines what action should be taken if more than one user attempt to modify the same data. When the value is set to `OverwriteChanges`, the control uses a *Last in Wins* style of updating data. In this style, the control overwrites any changes to data that have been made between the time the data was retrieved by the control and the time the update is made.

If the value is set to `CompareAllValues`, the data source control compares the original data values (what was retrieved) to the data values currently in the data store. If the data has not changed since it was retrieved, the control allows the changes to be implemented. If the control detects differences between the original data that was retrieved from the data store and what is currently in the data store, it does not allow the update to continue. This could potentially occur when you have multiple users accessing the data store and making changes to the data at the same time. In this case, another user could possibly retrieve and change the data well before you send your own changes to the data store. If you don't want to override the previous user's changes, you need to use the `CompareAllValues` value. Listing 7-5 shows how to add the `ConflictDetection` property to the SqlDataSource control.

Listing 7-5: Adding the ConflictDetection property to a SqlDataSource control

```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server"
   SelectCommand="SELECT * FROM [Customers] WHERE ([CustomerID] = @CustomerID)"
    ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>"
    DataSourceMode="DataSet"
    ConflictDetection="CompareAllValues">
    <SelectParameters>
        <asp:QueryStringParameter Name="CustomerID"
            QueryStringField="id" Type="String">
        </asp:QueryStringParameter>
    </SelectParameters>
</asp:SqlDataSource>
```

As described earlier, you can also use the SqlDataSource Configuration Wizard to add optimistic concurrency to the control. Doing this causes several changes in the underlying control. First, it automatically adds the `ConflictDetection` attribute to the control and sets it to `CompareAllValues`. Second, the wizard modifies the `Update` and `Delete` parameter collections to include parameters for the original data values. It also modifies the SQL statements so that they compare the original data values to the new values. You can recognize the newly added parameters because the wizard simply prepends the prefix `original` to each data column name. Listing 7-6 shows you what the modified `UpdateParameters` looks like.

Listing 7-6: Adding original value parameters to the UpdateParameters collection

```
<UpdateParameters>
    <asp:Parameter Name="CompanyName" Type="String" />
    <asp:Parameter Name="ContactName" Type="String" />
    <asp:Parameter Name="ContactTitle" Type="String" />
    <asp:Parameter Name="Address" Type="String" />
    <asp:Parameter Name="City" Type="String" />
    <asp:Parameter Name="Region" Type="String" />
    <asp:Parameter Name="PostalCode" Type="String" />
```

```
        <asp:Parameter Name="Country" Type="String" />
        <asp:Parameter Name="Phone" Type="String" />
        <asp:Parameter Name="Fax" Type="String" />
        <asp:Parameter Name="original_CustomerID" Type="String" />
        <asp:Parameter Name="original_CompanyName" Type="String" />
        <asp:Parameter Name="original_ContactName" Type="String" />
        <asp:Parameter Name="original_ContactTitle" Type="String" />
        <asp:Parameter Name="original_Address" Type="String" />
        <asp:Parameter Name="original_City" Type="String" />
        <asp:Parameter Name="original_Region" Type="String" />
        <asp:Parameter Name="original_PostalCode" Type="String" />
        <asp:Parameter Name="original_Country" Type="String" />
        <asp:Parameter Name="original_Phone" Type="String" />
        <asp:Parameter Name="original_Fax" Type="String" />
    </UpdateParameters>
```

Finally, the SqlDataSource Wizard sets an additional property called `OldValueParameterFormatString`. This attribute determines the prefix for the original data values. By default, the value is `original_0`, but you have complete control over this.

One way to determine whether your update has encountered a concurrency error is by testing the `AffectedRows` property in the SqlDataSource's `Updated` event. Listing 7-7 shows one way to do this.

Listing 7-7: Detecting concurrency errors after updating data

VB

```
Protected Sub SqlDataSource1_Updated(ByVal sender as Object, _
   ByVal e As System.Web.UI.WebControls.SqlDataSourceStatusEventArgs)

    If (e.AffectedRows > 0) Then
        Message.Text = "The record has been updated"
    Else
        Message.Text = "Possible concurrency violation"
    End If
End Sub
```

C#

```
protected void SqlDataSource1_Updated(object sender,
   SqlDataSourceStatusEventArgs e)
{
    if (e.AffectedRows > 0)
        Message.Text = "The record has been updated";
    else
        Message.Text = "Possible concurrency violation";
}
```

SqlDataSource Events

The SqlDataSource control provides a number of events that you can hook into to affect the behavior of the SqlDataSource control or to react to events that occur while the SqlDataSource control is executing. The control provides events that are raised before and after the Select, Insert, Update, and Delete commands are executed. You can use these events to alter the SQL command being sent to the data source by the control. You can cancel the operation or determine if an error has occurred while executing the SQL command.

Using the Data Source Events to Handle Database Errors

The data source control events are very useful for trapping and handling errors that occur while you are attempting to execute a SQL command against the database. For instance, Listing 7-8 demonstrates how you can use the SqlDataSource control's `Updated` event to handle a database error that has bubbled back to the application as an exception.

Listing 7-8: Using the SqlDataSource control's Updated event to handle database errors

VB

```
Protected Sub SqlDataSource1_Updated(ByVal sender As Object, _
    ByVal e As System.Web.UI.WebControls.SqlDataSourceStatusEventArgs)

    If (e.Exception IsNot Nothing) Then
        'An exception has occured executing the
        ' SQL command and needs to be handled
        lblError.Text = e.Exception.Message

        'Finally, tell ASP.NET you have handled the
        ' exception and it is to continue
        ' executing the application code
        e.ExceptionHandled = True
    End If
End Sub
```

C#

```
protected void SqlDataSource1_Updated(object sender,
    System.Web.UI.WebControls.SqlDataSourceStatusEventArgs e)
{
    if (e.Exception != null)
    {
        //An exception has occured executing the
        // SQL command and needs to be handled
        lblError.Text = e.Exception.Message;

        //Finally, tell ASP.NET you have handled the
        // exception and it is to continue
        // executing the application code
        e.ExceptionHandled = true;
    }
}
```

Notice that the sample tests to see if the `Exception` property is `null`; if it is not (indicating an exception has occurred), the application attempts to handle the exception.

An extremely important part of this sample is the code that sets the `ExceptionHandled` property. By default, this property returns `False`. Therefore, even if you detect the `Exception` property is not null and you attempt to handle the error, the exception still bubbles out of the application. Setting the `ExceptionHandled` property to `True` tells .NET that you have successfully handled the exception and that it is safe to continue executing. Note that the `AccessDataSource` and `ObjectDataSource` controls also function in this manner.

Although the SqlDataSource control is powerful, a number of other data source controls might suit your specific data access scenario better.

Using the SqlDataSource with Oracle

Just as you would use the SqlDataSource control to connect to Microsoft's SQL Server, you can also use this same control to connect to other databases that might be contained within your enterprise, such as Oracle.

To use the SqlDataSource control with Oracle, start by dragging and dropping the SqlDataSource control onto your page's design surface. Using the SqlDataSource control's smart tag, you are then able to configure your data source by clicking the Configure Data Source link.

Within the Configure Data Source wizard, you first need to create a new connection to your Oracle database. Click on the New Connection button to open the Add Connection dialog which is shown in Figure 7-6.

By default, the Data Source is configured to work with a SQL Server database but you can change this by simply pressing the Change button. This will launch a new dialog that allows you to select Oracle. This dialog is presented here in Figure 7-7.

> **This discussion is for the Microsoft Oracle Provider that comes with Visual Studio. Oracle also has its own provider, named the Oracle Data Provider (ODP), and the procedures to use that would be different. Consult the ODP documentation for details.**

Selecting an Oracle database will then modify the Add Connection dialog so that it is more appropriate for the job. This is presented in Figure 7-8.

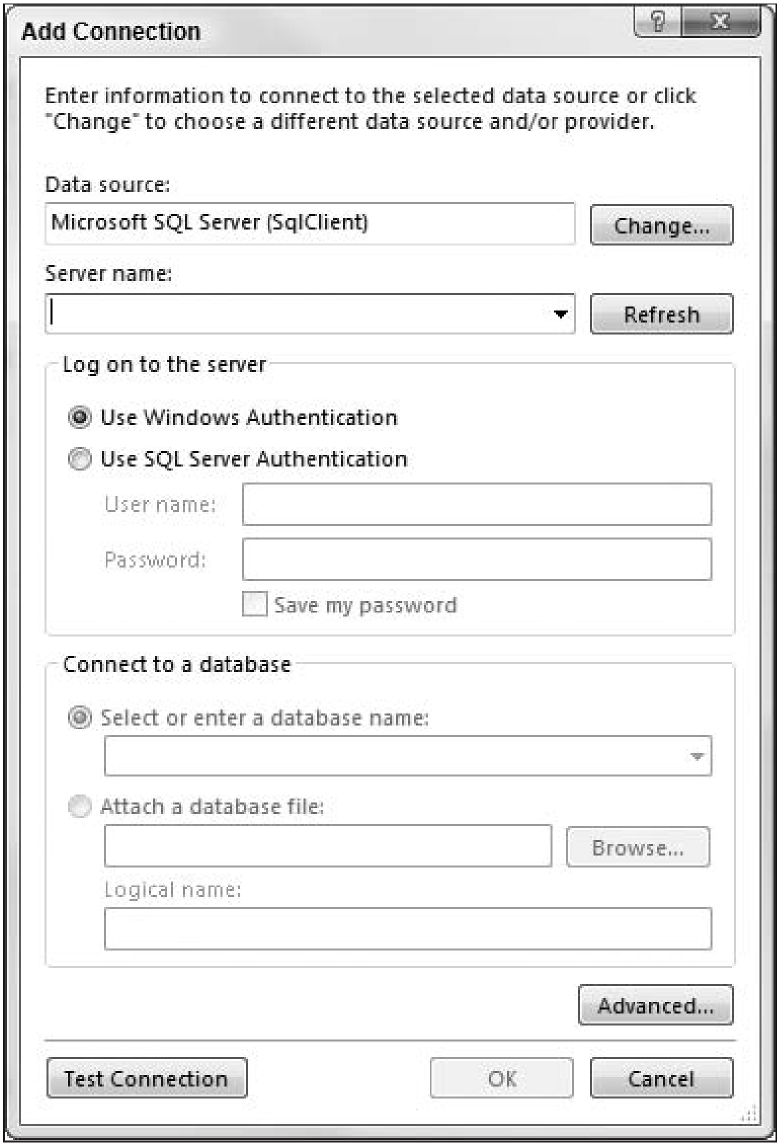
From the Add Connection dialog, you can add the name of the Oracle database that you are connecting to in the Server name text box. The name you place here is the name of the database that is held in the `tnsnames.ora` configuration file. This file is put into place after you install the Oracle Client on the server that will be making the connection to Oracle. You will find this file under a folder structure that is specific to the version of the Oracle client software — for example: `C:\Oracle\product\10.1.0\Client_1\NETWORK\ADMIN`. One particular database entry in this `tnsnames.ora` file is presented here:

```
MyDatabase =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.101)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SID = MyDatabase)
      (SERVER = DEDICATED)
    )
  )
```

After the reference to the database, you can then use your database username and password and then simply use the SqlDataSource control as you would if you were working with SQL Server. Once the configuring of the SqlDataSource is complete, you will then find a new connection to Oracle in your

`<connectionStrings>` section of the `web.config` (if you chose to save the connection string there through the configuration process). An example of this is presented here:

```
<connectionStrings>
  <add name="ConnectionString"
   connectionString="Data Source=MyDatabase;User
      ID=user1;Password=admin1pass;Unicode=True"
   providerName="System.Data.OracleClient" />
 </connectionStrings>
```



Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient)

Log on to the server

● Use Windows Authentication

○ Use SQL Server Authentication

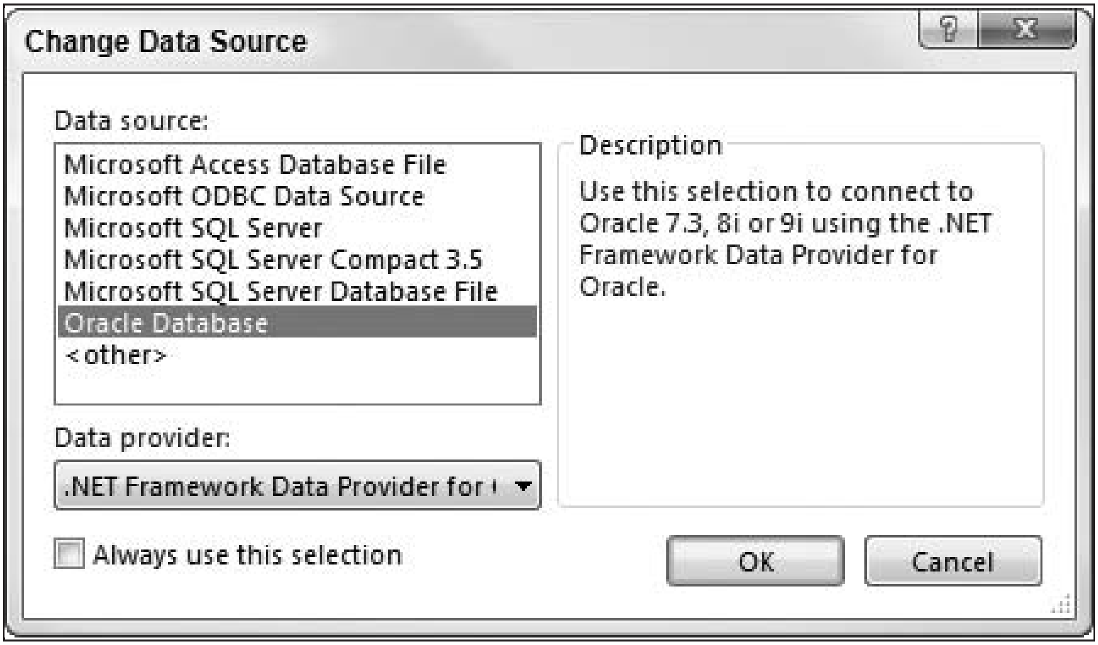
Connect to a database

● Select or enter a database name:

○ Attach a database file:

Change...
Server name:
Refresh
User name:
Password:
☐ Save my password
Browse...
Logical name:
Advanced...
Test Connection
OK
Cancel

Figure 7-6

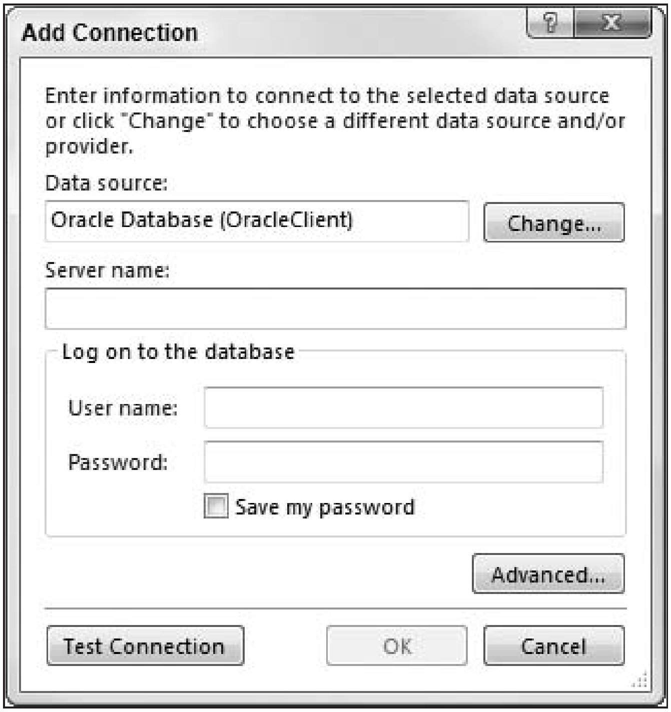

Change Data Source
Data source:
Microsoft Access Database File
Microsoft ODBC Data Source
Microsoft SQL Server
Microsoft SQL Server Compact 3.5
Microsoft SQL Server Database File
<other>
Description
Data provider:
☐ Always use this selection
OK
Cancel

Data source: Oracle Database

Data provider: .NET Framework Data Provider for (

Use this selection to connect to Oracle 7.3, 8i or 9i using the .NET Framework Data Provider for Oracle.

Figure 7-7


Add Connection
Change...
Server name:
Log on to the database
User name:
Password:
☐ Save my password
Advanced...
Test Connection
OK
Cancel

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Oracle Database (OracleClient)

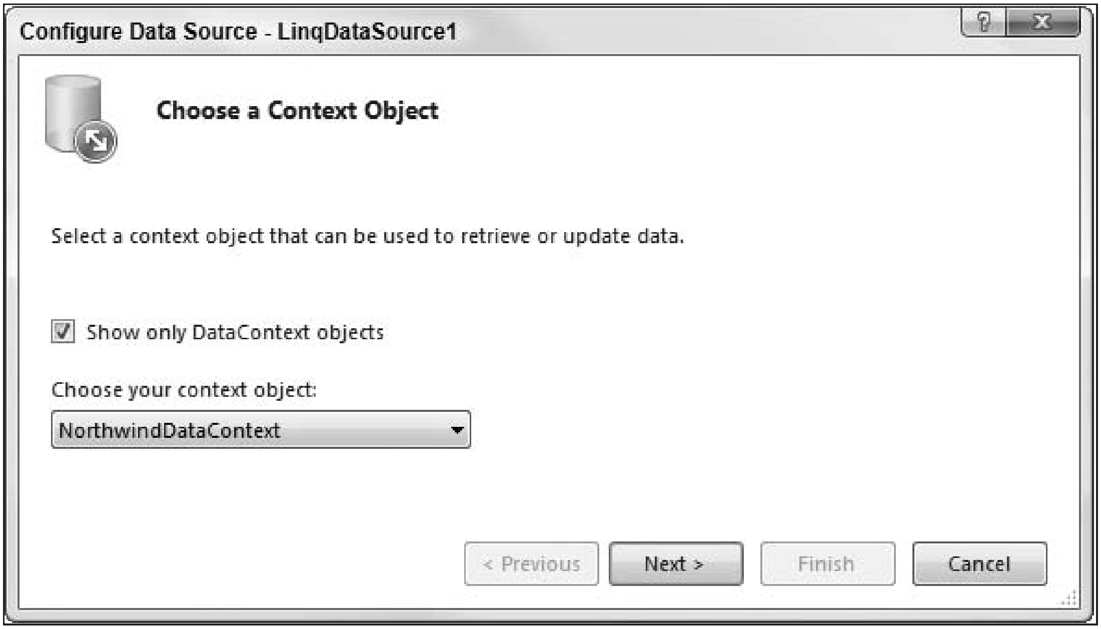
Figure 7-8

LINQ Data Source Control

The LinqDataSource control is a new Data Source control introduced in ASP.NET 3.5 that allows you to use the new LINQ features of .NET 3.5 to query data objects in your application.

> *This chapter focuses primarily on how to use the LinqDataSource control and its design-time configuration options. If you want to learn more about LINQ, its syntax, and how it works with different object types, refer to Chapter 9 in this book.*

The LinqDataSource control works much the same way as any other Data Source control, converting the properties you set on the control into queries that can be executed on the targeted data object. Much like the SqlDataSource control, which generated SQL statements based on your property settings, the LinqDataSource control converts the property settings into valid LINQ queries. When you drag the control onto the Visual Studio design surface, you can use the smart tag to configure the control. Figure 7-9 shows the initial screen of the configuration wizard.



Select a context object that can be used to retrieve or update data.

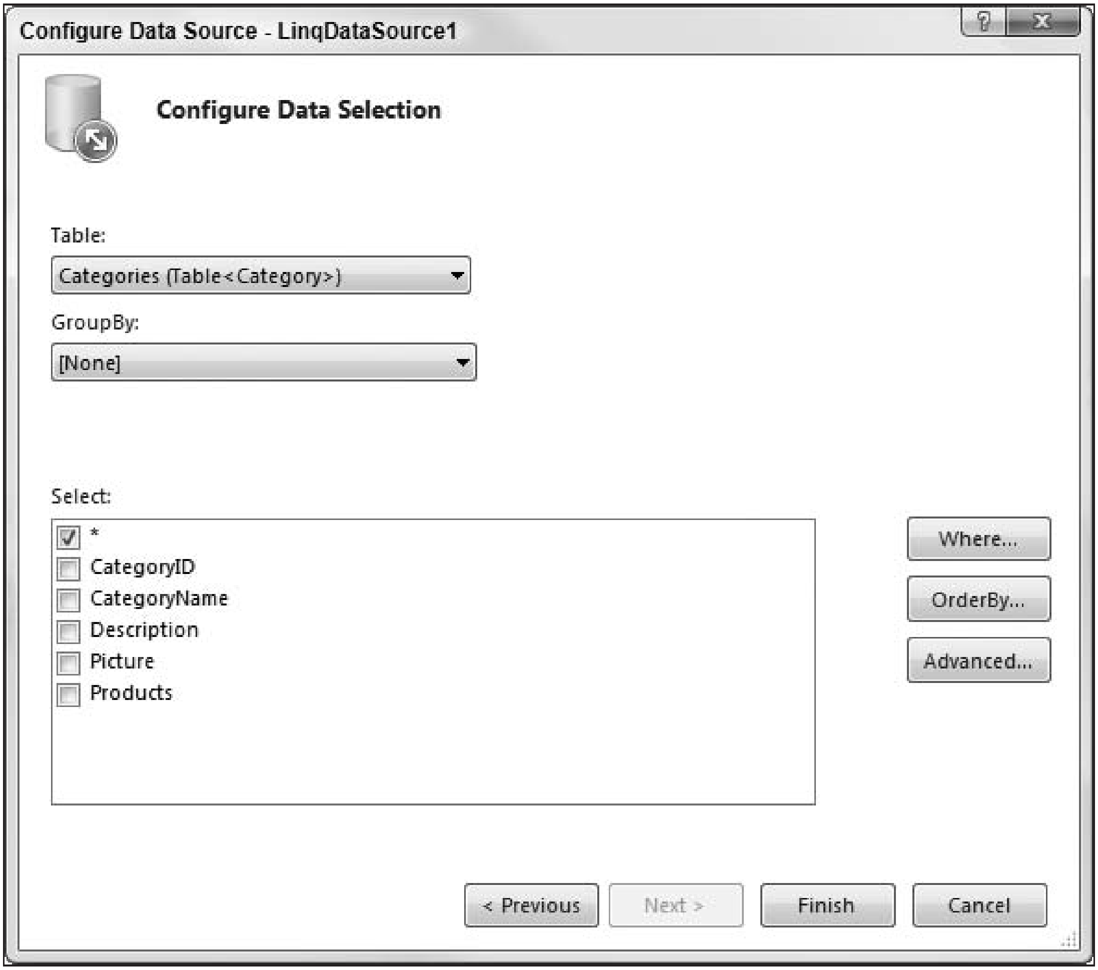
☑ Show only DataContext objects

Choose your context object: NorthwindDataContext

Figure 7-9

From this screen you can choose the context object you want to use as the source of your data. The context object is the base object that contains the data you want to query. By default, the wizard will show only objects that are derived from the `System.Data.Linq.DataContext` base class, which are normally data context classes created by LINQ to SQL. The wizard does give you the option of seeing all objects in your application (even those included as references in your project) and allowing you to select one of those as your context object.

Once you have selected your context object, the wizard allows you to select the specific table or property within the context object that returns the data you want to bind to, as shown in Figure 7-10. If you are binding to a class derived from `DataContext`, the table drop-down list shows all of the data tables contained in the context object. If you are binding to a standard class, then the drop-down allows you to select any enumerable property exposed by the context object.



Configure Data Source - LinqDataSource1
Configure Data Selection
Table:
Categories (Table<Category>)
GroupBy:
[None]
Select:
☑ *
☐ CategoryID
☐ CategoryName
☐ Description
☐ Picture
☐ Products
Where...
OrderBy...
Advanced...
< Previous
Next >
Finish
Cancel

Figure 7-10

Once you have selected the table, you can click the Finish button and complete the wizard. Listing 7-9 shows the markup that is generated by the LinqDataSource Configuration Wizard after it is configured to use the Northwind database as its context object and the Customers table.

Listing 7-9: The basic LinqDataSource control markup

```
<asp:LinqDataSource ID="LinqDataSource1" runat="server"
    ContextTypeName="NorthwindDataContext" TableName="Customers"
    EnableInsert="True" EnableUpdate="True" EnableDelete="True">
</asp:LinqDataSource>
```

The LinqDataSource is now ready to be bound to a data control such as a GridView or ListView.

Notice that the markup generated by the control includes three properties: `EnableInsert`, `EnableUpdate` and `EnableDelete`. These properties allow you to configure the control to allow Insert, Update and Delete actions if the underlying data source supports them. Because the data source control knows that it is connected to a LINQ To SQL data context object, which by default supports these actions, it has automatically enabled them.

The LinqDataSource also includes a number of other basic configuration options you can use to control the selection of data from the context object. As shown in Figure 7-10, the configuration wizard also allows you to select specific fields to include in its resultset.

> *While this can be a convenient way to control which fields are displayed in a bound control such as the GridView, it also causes the underlying LINQ query to return a custom projection. A side effect of this is that the resulting dataset no longer supports the inserting, updating, or deletion of data. If you simply want to limit the data shown by the bound list control, you may want to consider defining the fields to display in the bound list control rather than in the data source control.*

If you choose to select specific fields for the LinqDataSource control to return, the wizard adds the `Select` attribute to the markup with the appropriate LINQ projection statement. This is shown in Listing 7-10, where the control has been modified to return only the CustomerID, ContactName, ContactTitle, and Region fields.

Listing 7-10: Specifying LinqDataSource control data fields

```
<asp:LinqDataSource ID="LinqDataSource1" runat="server"
    ContextTypeName="NorthwindDataContext" TableName="Customers"
    Select="new (CustomerID, ContactName, ContactTitle, Region)">
</asp:LinqDataSource>
```

Binding the control to the GridView, you will now see that only these four specified fields are displayed. If no `Select` property is specified, the LinqDataSource control simply returns all public properties exposed by the data object.

Query Operations

The LinqDataSource control also allows you to specify different query parameters such as `Where` and `OrderBy`. Configuration of either option is available by clicking the Where or OrderBy buttons in the Controls Configuration Wizard.

Defining a Where Clause

The `Where` parameters are created using the same basic `Parameters` syntax used by other Data Source controls, which means that you can provide values from a variety of runtime sources such as Form fields, Querystring values, or even Session values. Listing 7-11 demonstrates the use of `Where` parameters.

Listing 7-11: Specifying Where clause parameters

```
<asp:LinqDataSource ID="LinqDataSource1" runat="server"
    ContextTypeName="NorthwindDataContext" TableName="Customers"
    Select="new (CustomerID, ContactName, ContactTitle, Region)"
    Where="CustomerID == @CustomerID">
    <whereparameters>
```

```
        <asp:querystringparameter DefaultValue="0" Name="CustomerID"
            QueryStringField="cid" Type="String" />
    </whereparameters>
</asp:LinqDataSource>
```

You can add multiple `Where` parameters that the control will automatically concatenate in its `Where` property using the `AND` operator. You can manually change the default value of the `Where` property if you want to have multiple `WhereParameters` defined, but only use a subset, or if you want to change which parameters are used dynamically at runtime. This is shown in Listing 7-12, where the LinqDataSource control has several `Where` parameters defined, but by default is using only one.

Listing 7-12: Using one of multiple defined WhereParameters

```
<asp:LinqDataSource ID="LinqDataSource1" runat="server"
    ContextTypeName="NorthwindDataContext" TableName="Customers"
    Select="new (CustomerID, ContactName, ContactTitle, Region)"
    Where="Country == @Country"
        OrderBy="Region, ContactTitle, ContactName">
    <whereparameters>
        <asp:querystringparameter DefaultValue="0" Name="CustomerID"
            QueryStringField="cid" Type="String" />
        <asp:querystringparameter DefaultValue="USA" Name="Country"
            QueryStringField="country" Type="String" />
        <asp:FormParameter DefaultValue="AZ" Name="Region"
            FormField="region" Type="String" />
     </whereparameters>
</asp:LinqDataSource>
```

In this case, although three `WhereParameters` are defined, the `Where` property uses only the `Country` parameter. It would be simple to change the `Where` property's value at runtime to use any of the defined parameters, based perhaps on a configuration setting set by the end user.

The LinqDataSource control also includes a property called `AutoGenerateWhereClause`, which can simplify the markup created by the control. When set to `True`, the property causes the control to ignore the value of the `Where` property and automatically use each parameter specified in the `Where` parameters collection in the query's `Where` clause.

Defining an OrderBy Clause

When defining an `OrderBy` clause, by default the wizard simply creates a comma-delimited list of fields as the value for the control's `OrderBy` property. The value of the `OrderBy` property is then appended to the control's LINQ query when it is executed.

The control also exposes an `OrderByParameters` collection, which you can also use to specify `OrderBy` values. However, in most cases using the simple `OrderBy` property will be sufficient. You would need to the use `OrderBy` parameters collection only if you need to determine the value of a variable at runtime, and then order the query results based on that value.

Grouping Query Data

The LinqDataSource control also makes it easy to specify a grouping for the resultset returned by the query. In the configuration wizard, you can select the Group By field for the query. Once a field is selected, the wizard then creates a default projection based on the groupby field. The projection includes two fields by default, the first called `key`, which represents the group objects specified in the `GroupBy` property, and the second called `it` which represents the grouped objects. You can also add your own columns to the projection and execute functions against the grouped data such as `Average`, `Min`, `Max`, and `Count`. Listing 7-13 demonstrates a very simple grouping using the LinqDataSource control.

Listing 7-13: Simple Data Grouping

```
<asp:LinqDataSource ID="LinqDataSource1" runat="server"
   ContextTypeName="NorthwindDataContext" TableName="Products"
    Select="new (key as Category, it as Products,
        Average(UnitPrice) as Average_UnitPrice)"
    GroupBy="Category">
</asp:LinqDataSource>
```

You can see in this sample that the Products table has been grouped by its `Category` property. The LinqDataSource control created a new projection containing the `key` (aliased as `Category`) and `it` (aliased as `Products`) fields. Additionally, the custom field `average_unitprice` has been added, which calculates the average unit price of the products in each category. When you execute this query and bind the results to a GridView, a simple view of the calculated average unit price is displayed. The Category and Products are not displayed because they are complex object types, and the GridView is not capable of directly displaying them.

You can access the data in either key by using the standard `ASP.NET Eval()` function. In a GridView you would do this by creating a `TemplateField`, and using `ItemTemplate` to insert the `Eval` statement as shown here:

```
<asp:TemplateField>
    <ItemTemplate>
        <%# Eval("Category.CategoryName") %>
    </ItemTemplate>
</asp:TemplateField>
```

In this case, `TemplateField` displays the `CategoryName` for each grouped category in the resultset.

Accessing the grouped items is just as easy. If, for example, you wanted to include a bullet list of each product in an individual group, you would simply add another `TemplateField`, insert a BulletList control and bind it to the Products field returned by the query, as shown next:

```
<asp:TemplateField>
    <ItemTemplate>
        <asp:BulletedList DataSource='<%# Eval("Products") %>'
            DataTextField="ProductName" runat="server" ID="BulletedList" />
    </ItemTemplate>
</asp:TemplateField>
```

Data Concurrency

Like the SqlDataSource control, the LinqDataSource control allows for data concurrency checks when updating or deleting data. As its name implies, the `StoreOriginalValuesInViewState` property indicates whether the data source control should store the original data values in ViewState. Doing this when using LINQ to SQL as the underlying data object, allows LINQ to SQL to perform data concurrency checking before submitting updates, or deleting data.

Storing the original data in ViewState, however, can cause the size of your Web page to grow significantly, affecting the performance of your Web site, so you may wish to disable the storing of data in ViewState. If you do, recognize that you are now responsible for any data concurrency checking that your application requires.

LinqDataSource Events

The LinqDataSource control also includes a number of useful events that you can use to react to actions taken by the control at runtime. Standard before and after events for Select, Insert, Update, and Delete actions are all exposed and allow you to add, remove, or modify parameters from the control's various parameter collections, or even cancel the event entirely.

Additionally, the post action events allow you to determine whether an exception has occurred while attempting to execute an Insert, Update, or Delete. If an exception has occurred, these events allow you to react to those exceptions, and either mark the exception as handled, or allow it to continue to bubble up through the application.

AccessDataSource Control

Although you can use the SqlDataSource to connect to Access databases, ASP.NET also provides a special `AccessDataSource` control. This control gives you specialized access to Access databases using the Jet Data provider, but it still uses SQL commands to perform data retrieval because it is derived from the SqlDataSource.

Despite its relative similarity to the SqlDataSource control, the AccessDataSource control has some specialized parts. First, the control does not require you to set a `ConnectionString` property. Instead, the control uses a `DataFile` property to allow you to directly specify the Access `.mdb` file you want to use for data access.

> *A side effect of not having the* `ConnectionString` *property is that the AccessDataSource cannot connect to password-protected databases. If you need to access a password-protected Access database, you can use the SqlDataSource control, which allows you to provide the username and password as part of the connection string.*

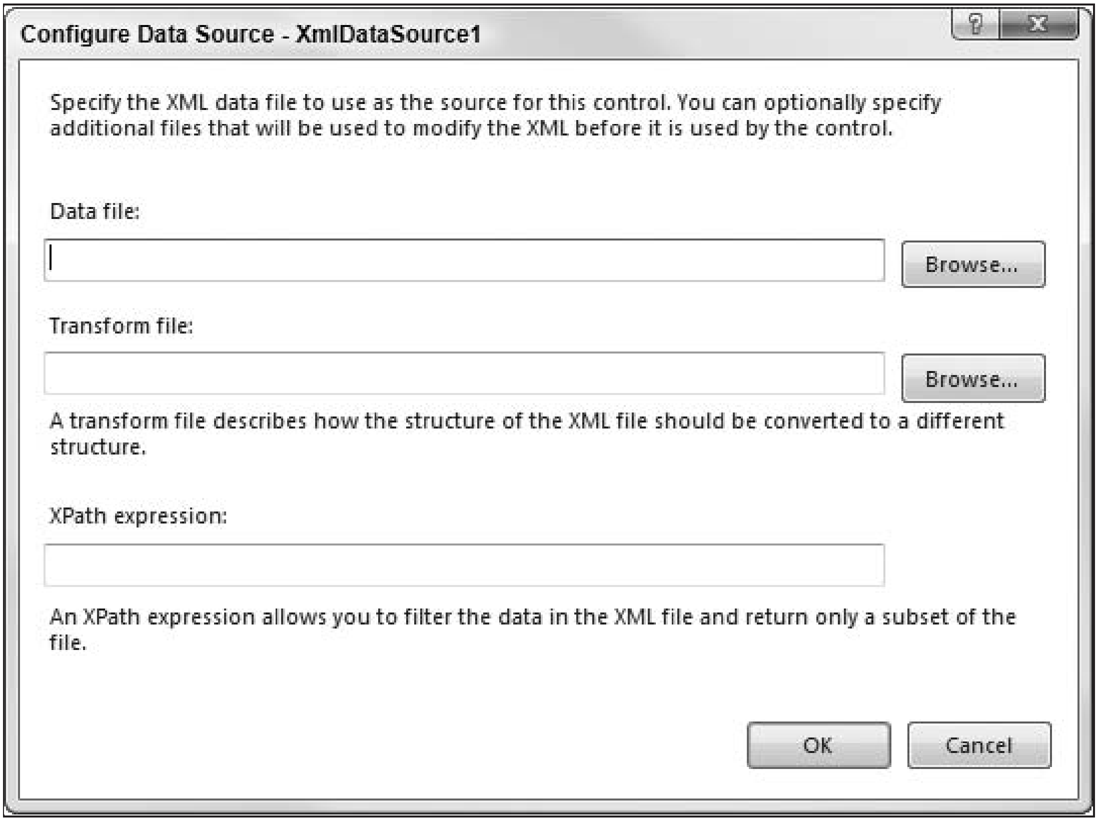
Additionally, because the AccessDataSource uses the System.Data.OleDb to perform actual data access, the order of parameters matters. You need to verify that the order of the parameters in any Select, Insert, Update, or Delete parameters collection matches the order of the parameters in the SQL statement.

XmlDataSource Control

The `XmlDataSource` control provides you with a simple way of binding XML documents, either in-memory or located on a physical drive. The control provides you with a number of properties that make it easy to specify an XML file containing data and an XSLT transform file for converting the source XML into a more suitable format. You can also provide an XPath query to select only a certain subset of data.

You can use the XmlDataSource control's Configure Data Wizard, shown in Figure 7-11, to configure the control.

Listing 7-14 shows how you might consume an RSS feed from the MSDN Web site, selecting all the item nodes within it for binding to a bound list control such as the GridView.



Configure Data Source - XmlDataSource1

Specify the XML data file to use as the source for this control. You can optionally specify additional files that will be used to modify the XML before it is used by the control.

Data file:

Transform file:

A transform file describes how the structure of the XML file should be converted to a different structure.

XPath expression:

An XPath expression allows you to filter the data in the XML file and return only a subset of the file.

Figure 7-11

Listing 7-14: Using the XmlDataSource control to consume an RSS feed

```
<asp:XmlDataSource ID="XmlDataSource1" Runat="server"
    DataFile="http://msdn.microsoft.com/rss.xml"
    XPath="rss/channel/item"
</asp:XmlDataSource>
```

In addition to the declarative attributes you can use with the XmlDataSource, a number of other helpful properties and events are available.

Many times, your XML is not stored in a physical file, but rather is simply a string stored in your application memory or possibly in a SQL database. The control provides the `Data` property, which accepts a simple string of XML to which the control can bind. Note that if both the `Data` and `DataFile` properties are set, the `DataFile` property takes precedence over the `Data` property.

Additionally, in certain scenarios, you may want to export the bound XML out of the XmlDataSource control to other objects or even save any changes that have been made to the underlying XML if it has been bound to a control such as a GridView. The XmlDataSource control provides two methods to accommodate this. First, the `GetXmlDocument` method allows you to export the XML by returning a basic `System.Xml.XmlDocument` object that contains the XML loaded in the data source control.

Second, using the control's `Save` method, you can persist changes made to the XmlDataSource control's loaded XML back to disk. Executing this method assumes you have provided a file path in the `DataFile` property.

The XmlDataSource control also provides you with a number of specialized events. The `Transforming` event that is raised before the XSLT transform specified in the `Transform` or `TransformFile` properties is applied and allows you to supply custom arguments to the transform.

ObjectDataSource Control

The `ObjectDataSource` control is one of the most interesting data source controls in the ASP.NET toolbox. It gives you the power to bind data controls to middle-layer business objects that can be hard-coded or automatically generated from programs such as Object Relational (O/R) mappers.

To demonstrate how to use the ObjectDataSource control, create a class in the project that represents a customer. Listing 7-15 shows a class that you can use for this demonstration.

Listing 7-15: Creating a Customer class to demonstrate the ObjectDataSource control

VB

```
Public Class Customer
     Private _customerID As Integer
     Private _companyName As String
     Private _contactName As String
     Private _contactTitle As String

     Public Property CustomerID() As Integer
          Get
               Return _customerID
          End Get
          Set
               _customerID = value
          End Set
     End Property

     Public Property CompanyName() As Integer
          Get
               Return _companyName
          End Get
          Set
               _companyName = value
          End Set
     End Property

     Public Property ContactName() As Integer
          Get
               Return _contactName
          End Get
          Set
               _contactName = value
          End Set
```

Continued

```
    End Property

    Public Property ContactTitle() As Integer
        Get
            Return _contactTitle
        End Get
        Set
            _contactTitle = value
        End Set
    End Property

    Public Function [Select](ByVal customerID As Integer) As System.Data.DataSet
        ' You would implement logic here to retreive
        ' Customer data based on the customerID parameter

      Dim ds As New System.Data.DataSet()
      ds.Tables.Add(New System.Data.DataTable())
      Return ds
    End Function

    Public Sub Insert(ByVal c As Customer)
        ' Implement Insert logic
    End Sub

    Public Sub Update(ByVal c As Customer)
        ' Implement Update logic
    End Sub

    Public Sub Delete(ByVal c As Customer)
        ' Implement Delete logic
    End Sub

End Class
```

C#

```
public class Customer
{
    private int _customerID;
    private string _companyName;
    private string _contactName;
    private string _contactTitle;

    public int CustomerID
    {
        get
        {
            return _customerID;
        }

        set
        {
            _customerID = value;
        }
    }
```

Continued

```
public string CompanyName
{
    get
    {
        return _companyName;
    }

    set
    {
        _companyName = value;
    }
}

public string ContactName
{
    get
    {
        return _contactName;
    }

    set
    {
        _contactName = value;
    }
}

public string ContactTitle
{
    get
    {
        return _contactTitle;
    }

    set
    {
        _contactTitle = value;
    }
}

public Customer()
{
}

public System.Data.DataSet Select(Int32 customerId)
{
    // Implement logic here to retrieve the Customer
    // data based on the methods customerId parameter

    System.Data.DataSet ds = new System.Data.DataSet();
    ds.Tables.Add(new System.Data.DataTable());
    return ds;
}
```

Continued

```
        public void Insert(Customer c)
        {
            // Implement Insert logic
        }

        public void Update(Customer c)
        {
            // Implement Update logic
        }

        public void Delete(Customer c)
        {
            // Implement Delete logic
        }

    }
```

To start using the ObjectDataSource, drag the control onto the designer surface. Using the control's smart tag, load the Configuration Wizard by selecting the Configure Data Source option. After the wizard opens, it asks you to select the business object you want to use as your data source. The drop-down list shows all the classes located in the `App_Code` folder of your Web site that can be successfully compiled. In this case, you want to use the `Customer` class shown in Listing 7-8.

Click the Next button, and the wizard asks you to specify which methods it should use for the CRUD operations it can perform: SELECT, INSERT, UPDATE, and DELETE. Each tab lets you select a specific method located in your business class to perform the specific action. Figure 7-12 shows that you want the control to use a method called `Select()` to retrieve data.

The methods the ObjectDataSource uses to perform CRUD operations must follow certain rules in order for the control to understand. For instance, the control's `SELECT` method must return a `DataSet`, `DataReader`, or a strongly typed collection. Each of the control's operation tabs explains what the control expects of the method you specify for it to use. Additionally, if a method does not conform to the rules that specific operation expects, it is not listed in the drop-down list on that tab.

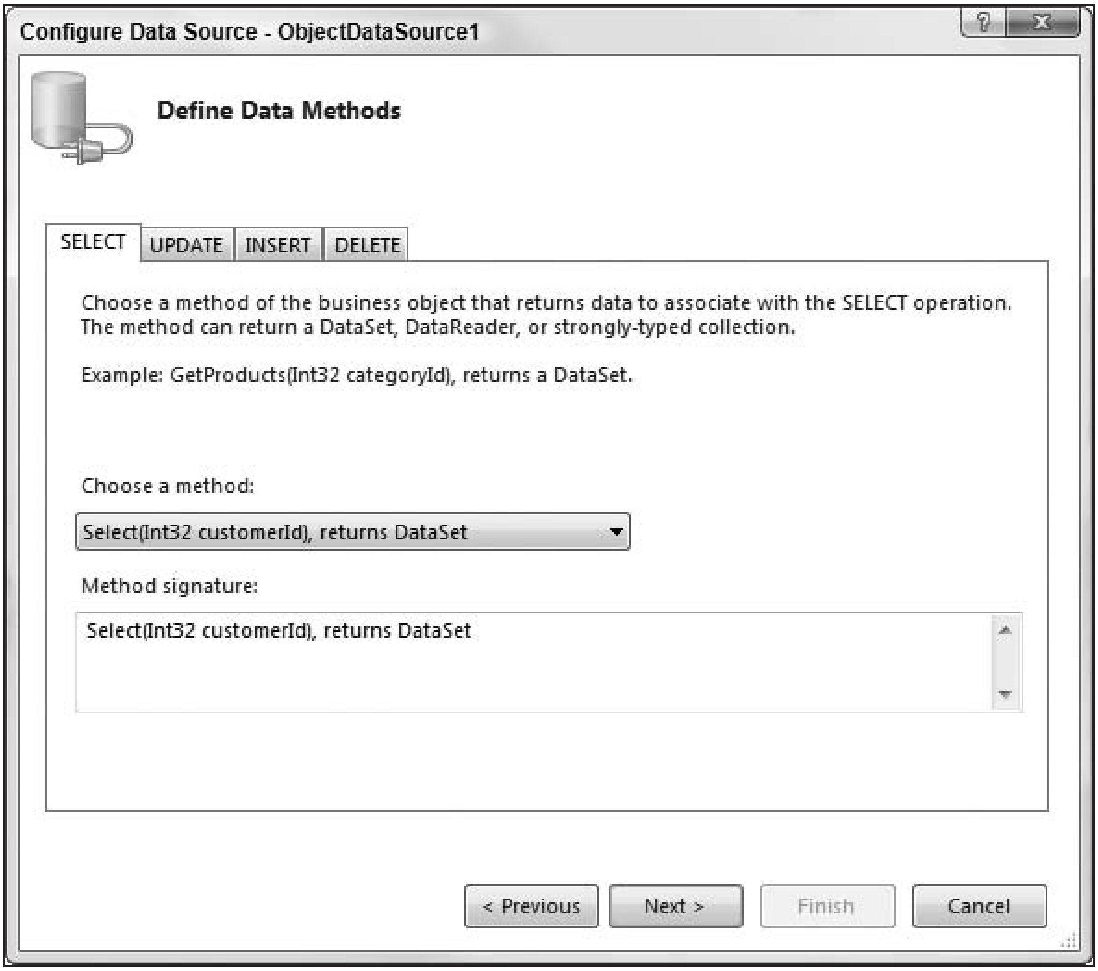
Finally, if your `SELECT` method contains parameters, the wizard lets you create `SelectParameters` you can use to provide the method parameter data.

When you have completed configuring the ObjectDataSource, you should have code in your page source like that shown in Listing 7-16.

Listing 7-16: The ObjectDataSource code generated by the configuration wizard

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server" DeleteMethod="Delete"
    InsertMethod="Insert" SelectMethod="Select" TypeName="Customer"
    UpdateMethod="Update">
    <SelectParameters>
        <asp:QueryStringParameter Name="customerID" QueryStringField="ID"
            Type="Int32" />
    </SelectParameters>
</asp:ObjectDataSource>
```

As you can see, the wizard has generated the attributes for the `SELECT`, `UPDATE`, `INSERT`, and `DELETE` methods you specified in the wizard. Also notice that it has added the `Select` parameter. Depending on your application, you could change this to any of the `Parameter` objects discussed earlier, such as a `ControlParameter` or `QuerystringParameter` object.



Choose a method of the business object that returns data to associate with the SELECT operation. The method can return a DataSet, DataReader, or strongly-typed collection.

Example: GetProducts(Int32 categoryId), returns a DataSet.

Choose a method:

Select(Int32 customerId), returns DataSet

Method signature:

Select(Int32 customerId), returns DataSet

Figure 7-12

ObjectDataSource Events

The ObjectDataSource control exposes several useful events that you can hook into. First, it includes events that are raised before and after the control performs any of the CRUD actions, such as selecting or deleting events.

It also includes pre- and post-events that are raised when the object that is serving as the data source is created or disposed of, as well as when an event that is raised before a filter is applied to the data. All these events give you great power when you must react to the different ways the ObjectDataSource control behaves.

SiteMapDataSource Control

The `SiteMapDataSource` enables you to work with data stored in your Web site's SiteMap configuration file if you have one. This can be useful if you are changing your site map data at runtime, perhaps based on user privilege or status.

Note two items regarding the SiteMapDataSource control. First, it does not support any of the data caching options that exist in the other data source controls provided (covered in the next section), so you cannot natively cache your sitemap data. Second, the SiteMapDataSource control does not have any configuration wizards like the other data source controls. This is because the SiteMap control can be bound only to the SiteMap configuration data file of your Web site, so no other configuration is possible.

Listing 7-17 shows an example of using the SiteMap control.

Listing 7-17: Using the SiteMapDataSource control

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" Runat="server" />
```

Using the SiteMapDataSource control is discussed in greater detail in Chapter 14.

Configuring Data Source Control Caching

Caching is built into all the data source controls that ship with ASP.NET except the SiteMapDataSource control. This means that you can easily configure and control data caching using the same declarative syntax. All data source controls (except the SiteMapDataSource control) enable you to create basic caching policies including a cache direction, expiration policies, and key dependencies.

> *Remember that the SqlDataSource control's caching features are available only if you have set the* `DataSourceMode` *property to* `DataSet`*. If it is set to* `DataReader`*, the control throws a* `NotSupportedException`*.*

Cache duration can be set to a specific length of time, such as 3600 seconds (60 minutes), or you can set it to `Infinite` to force the cached data to never expire. Listing 7-18 shows how you can easily add caching features to a data source control.

Listing 7-18: Enabling caching on a SqlDataSource control

```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server"
    SelectCommand="SELECT * FROM [Customers]"
    ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>"
    DataSourceMode="DataSet"
    ConflictDetection="CompareAllValues"
    EnableCaching="True" CacheKeyDependency="SomeKey" CacheDuration="Infinite">
    <SelectParameters>
        <asp:QueryStringParameter Name="CustomerID"
            QueryStringField="id" Type="String"></asp:QueryStringParameter>
    </SelectParameters>
</asp:SqlDataSource>
```

Some controls also extend this core set of caching features with additional caching functionality specific to their data sources. For instance, if you are using the SqlDataSource control, you can use the

`SqlCacheDependency` property to create SQL dependencies. You can learn more about ASP.NET caching features in Chapter 23.

Storing Connection Information

In ASP.NET 1.0/1.1, Microsoft introduced the `web.config` file as a way of storing application configuration data in a readable and portable format. Many people quickly decided that the `web.config` file was a great place to store things like the database connection information their applications use. It was easy to access from within the application, created a single central location for the configuration data, and it was a cinch to change just by editing the XML.

Although all those advantages were great, several drawbacks existed. First, none of the information in the `web.config` file can be strongly typed. It was, therefore, difficult to find data type problems within the application until a runtime error occurred. It also meant that developers were unable to use the power of IntelliSense to facilitate development. A second problem was that although the `web.config` file was secured from access by browsers (it cannot be served up by Internet Information Server), the data within the file was clearly visible to anyone who had file access to the Web server.

Microsoft has addressed these shortcomings of the `web.config` file by adding features specifically designed to make it easier to work with and secure connection string information in the `web.config` file. Because database connection information is so frequently stored in the `web.config` file, starting with ASP.NET 2.0, a new configuration section was added in that file. The `<connectionStrings>`section is designed specifically for storing the connection string information.

If you examine your `web.config` file, you should see at least one connection string already in the `<connectionStrings>` section because our example told the Data Connection Wizard to store connections in the `web.config` file. Listing 7-19 shows how ASP.NET stores a connection string.

Listing 7-19: A typical connection string saved in the web.config file

```
<connectionStrings>
    <add name="AppConnectionString1" connectionString="Server=localhost;
         User ID=sa;Password=password;Database=Northwind;
         Persist Security Info=True" providerName="System.Data.SqlClient" />
</connectionStrings>
```

Using a separate configuration section has several advantages. First, .NET exposes the `ConnectionString` section using the `ConnectionStringSettings` class. This class contains a collection of all the connection strings entered in your `web.config` file and allows you to add, modify, or remove connection strings at runtime. Listing 7-20 shows how you can access and modify connection strings at runtime.

Listing 7-20: Modifying connection string properties at runtime

VB

```
<%@ Page Language="VB" %>

<script runat="server">

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
```

Continued

```
        If (Not Page.IsPostBack) Then
            ' Create a new ConnectionStringSettings object and populate it
            Dim conn As New ConnectionStringSettings()
            conn.ConnectionString = _
                 "Server=localhost;User ID=sa;Password=password" & _
                 "Database=Northwind;Persist Security Info=True"
            conn.Name = "AppConnectionString1"
            conn.ProviderName = "System.Data.SqlClient"

            ' Add the new connection string to the web.config
            ConfigurationManager.ConnectionStrings.Add(conn)
        End If

    End Sub

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Modifying the Connection String</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:SqlDataSource ID="SqlDataSource1" Runat="server">
        </asp:SqlDataSource>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            // Create a new ConnectionStringSettings object and populate it
            ConnectionStringSettings conn = new ConnectionStringSettings();
            conn.ConnectionString =
                 "Server=localhost;User ID=sa;Password=password; " +
                 "Database=Northwind;Persist Security Info=True";
            conn.Name = "AppConnectionString1";
            conn.ProviderName = "System.Data.SqlClient";

            // Add the new connection string to the web.config
            ConfigurationManager.ConnectionStrings.Add(conn);
        }
    }
</script>
```

As you can see, the `ConfigurationManager` class now has a `ConnectionStrings` collection property in addition to the `AppSettings` collection used in ASP.NET 1.0. This new collection contains all the connection strings for your application.

Additionally, ASP.NET makes it much easier to build connection strings using strongly typed properties at runtime, and easier to add them to the `web.config` file. Using the new `SqlConnectionStringBuilder` class, you can build connection strings and then add them to your `ConnectionStringSettings` collection. Listing 7-21 shows how you can use the `ConnectionStringBuilder` class to dynamically assemble connection strings at runtime and save them to your `web.config` file.

Listing 7-21: Building connection strings using ConnectionStringBuilder

VB

```
' Retrieve an existing connection string into a Connection String Builder
Dim builder As New System.Data.SqlClient.SqlConnectionStringBuilder()

' Change the connection string properties
builder.DataSource = "localhost"
builder.InitialCatalog = "Northwind1"
builder.UserID = "sa"
builder.Password = "password"
builder.PersistSecurityInfo = true

' Save the connection string back to the web.config
ConfigurationManager.ConnectionStrings("AppConnectionString1").ConnectionString = _
    builder.ConnectionString
```

C#

```
// Retrieve an existing connection string into a Connection String Builder
System.Data.SqlClient.SqlConnectionStringBuilder builder = new
    System.Data.SqlClient.SqlConnectionStringBuilder();

// Change the connection string properties
builder.DataSource = "localhost";
builder.InitialCatalog = "Northwind1";
builder.UserID = "sa";
builder.Password = "password";
builder.PersistSecurityInfo = true;

// Save the connection string back to the web.config
ConfigurationManager.ConnectionStrings["AppConnectionString1"].ConnectionString =
    builder.ConnectionString;
```

Using Bound List Controls with Data Source Controls

The data source controls included in ASP.NET really shine when you combine them with the Bound List controls also included in ASP.NET. This combination allows you to declaratively bind your data source to a bound control without ever writing a single line of C# or VB code.

Fear not, those of you who like to write code. You can always use the familiar `DataBind()` *method to bind data to the list controls. In fact, that method has even been enhanced to include a Boolean overload that allows you to turn the data-binding events on or off. This enables you improve the performance of your application if you are not using any of the binding events.*

GridView

With ASP.NET 1.0/1.1, Microsoft introduced a new set of server controls designed to make developers more productive. One of the most popular controls was the DataGrid. With this one control, you could display an entire collection of data, add sorting and paging, and perform inline editing. Although this new functionality was great, many tasks still required that the developer write a significant amount of code to take advantage of this advanced functionality.

Beginning with ASP.NET 2.0, Microsoft took the basic DataGrid and enhanced it, creating a new server control called the GridView. This control makes it even easier to use those advanced DataGrid features, usually without having to write any code, and it also adds a number of new features.

Displaying Data with the GridView

Start using the GridView by dragging the control onto the designer surface of an ASP.NET Web page. You are prompted to select a data source control to bind to the grid. In this sample, you use the SqlDataSource control created earlier in the chapter.

After you assign the GridView a data source, notice a number of changes. First, the GridView changes its design-time display to reflect the data exposed by the data source control assigned to it. Should the schema of the data behind the data source control ever change, you can use the GridView's Refresh Schema option to force the grid to redraw itself based on the new data schema. Second, the GridView's smart tag has additional options for formatting, paging, sorting, and selection.

Switch the page to Source view in Visual Studio to examine GridView's code. Listing 7-22 shows the code generated by Visual Studio.

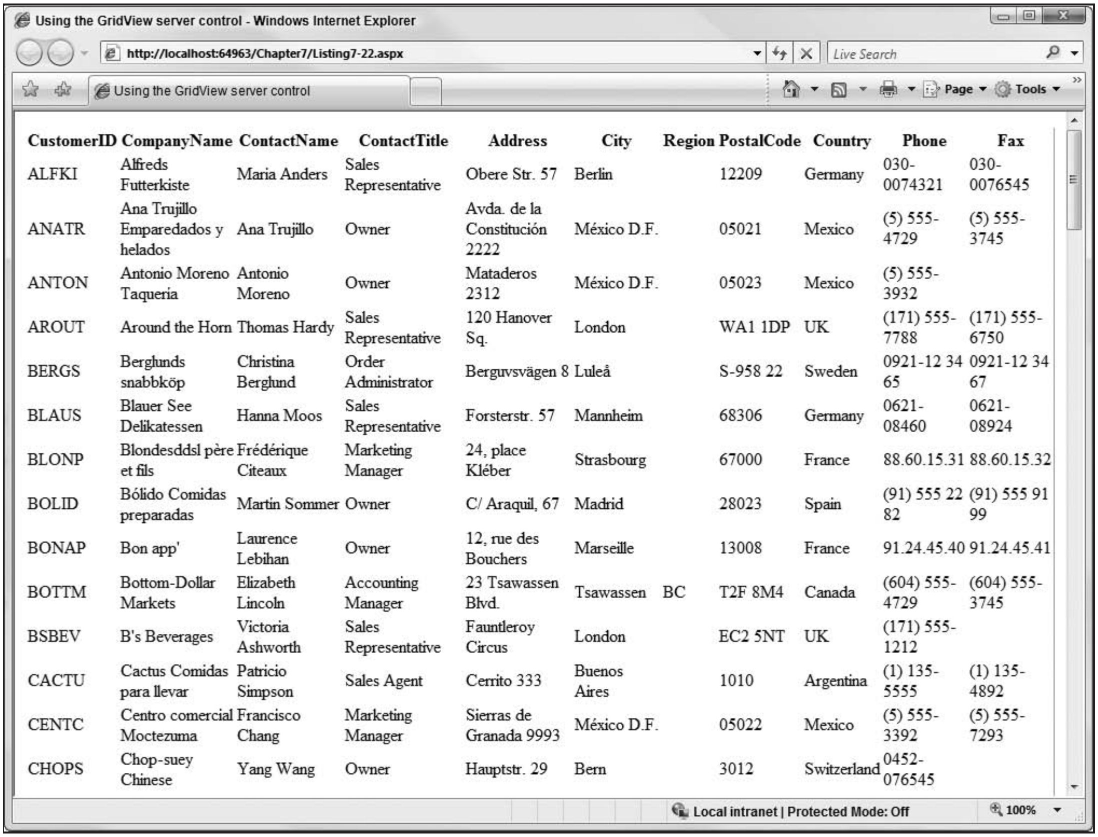
Listing 7-22: Using the GridView control in an ASP.NET Web page

```
<html>
<head runat="server">
    <title>Using the GridView server control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
          <asp:GridView ID="GridView1" Runat="server" DataSourceID="SqlDataSource1"
              DataKeyNames="CustomerID" AutoGenerateColumns="False">
            <Columns>
                <asp:BoundField ReadOnly="True" HeaderText="CustomerID"
                     DataField="CustomerID"
                     SortExpression="CustomerID"></asp:BoundField>
                <asp:BoundField HeaderText="CompanyName" DataField="CompanyName"
                     SortExpression="CompanyName"></asp:BoundField>
                <asp:BoundField HeaderText="ContactName" DataField="ContactName"
                     SortExpression="ContactName"></asp:BoundField>
                <asp:BoundField HeaderText="ContactTitle" DataField="ContactTitle"
                     SortExpression="ContactTitle"></asp:BoundField>
                <asp:BoundField HeaderText="Address" DataField="Address"
                     SortExpression="Address"></asp:BoundField>
                <asp:BoundField HeaderText="City" DataField="City"
                     SortExpression="City"></asp:BoundField>
                <asp:BoundField HeaderText="Region" DataField="Region"
                     SortExpression="Region"></asp:BoundField>
                <asp:BoundField HeaderText="PostalCode" DataField="PostalCode"
                     SortExpression="PostalCode"></asp:BoundField>
                <asp:BoundField HeaderText="Country" DataField="Country"
                     SortExpression="Country"></asp:BoundField>
                <asp:BoundField HeaderText="Phone" DataField="Phone"
                     SortExpression="Phone"></asp:BoundField>
                <asp:BoundField HeaderText="Fax" DataField="Fax"
                     SortExpression="Fax"></asp:BoundField>
            </Columns>
        </asp:GridView>

        <asp:SqlDataSource ID="SqlDataSource1" Runat="server"
            SelectCommand="SELECT * FROM [Customers]"
            ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>"
            DataSourceMode="DataSet"
            ConflictDetection="CompareAllValues" EnableCaching="True"
            CacheKeyDependency="MyKey" CacheDuration="Infinite">
        </asp:SqlDataSource>
    </div>
    </form>
</body>
</html>
```

Figure 7-13 shows what your Web page looks like when you execute the code in the browser.

The GridView includes several events that are raised when the data binding occurs. These are described in the following table.

Event Name	**Description**
`DataBinding`	Raised as the GridViews data-binding expressions are about to be evaluated
`RowCreated`	Raised each time a new row is created in the grid. Before the grid can be rendered, a `GridViewRow` object must be created for each row in the control. The `RowCreated` event allows you to insert custom content into the row as it is being created.
`RowDataBound`	Raised as each `GridViewRow` is bound to the corresponding data in the data source. This event allows you to evaluate the data being bound to the current row and to affect the output if you need to.
`DataBound`	Raised after the binding is completed and the GridView is ready to be rendered.


Using the GridView server control - Windows Internet Explorer
http://localhost:64963/Chapter7/Listing7-22.aspx
Live Search
Using the GridView server control
Page
Tools
Local intranet | Protected Mode: Off
100%

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076545
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK	(171) 555-7788	(171) 555-6750
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65	0921-12 34 67
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany	0621-08460	0621-08924
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France	88.60.15.31	88.60.15.32
BOLID	Bólido Comidas preparadas	Martin Sommer	Owner	C/ Araquil, 67	Madrid		28023	Spain	(91) 555 22 82	(91) 555 91 99
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille		13008	France	91.24.45.40	91.24.45.41
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen	BC	T2F 8M4	Canada	(604) 555-4729	(604) 555-3745
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative	Fauntleroy Circus	London		EC2 5NT	UK	(171) 555-1212	
CACTU	Cactus Comidas para llevar	Patricio Simpson	Sales Agent	Cerrito 333	Buenos Aires		1010	Argentina	(1) 135-5555	(1) 135-4892
CENTC	Centro comercial Moctezuma	Francisco Chang	Marketing Manager	Sierras de Granada 9993	México D.F.		05022	Mexico	(5) 555-3392	(5) 555-7293
CHOPS	Chop-suey Chinese	Yang Wang	Owner	Hauptstr. 29	Bern		3012	Switzerland	0452-076545	

Figure 7-13

The `RowDataBound` event is especially useful, enabling you to inject logic into the binding process for each row being bound. Listing 7-23 shows how you can use the `RowDataBound` to examine the data being bound to the current row and to insert your own logic. The code checks to see if the databases Region column is `null`; if it is, the code changes the `ForeColor` and `BackColor` properties of the GridView rows.

Listing 7-23: Using the RowDataBound to examine the data being bound to the current row and to insert your own logic

VB

```
<script runat="server">
    Protected Sub GridView1_RowDataBound(ByVal sender As Object, _
        ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs)

        'Test for null since gridview rows like the
        ' Header and Footer will have a null DataItem
        If (e.Row.DataItem IsNot Nothing) Then
            'Used to verify the DataItem object type
            System.Diagnostics.Debug.WriteLine(e.Row.DataItem.ToString())
            'When bound to a SqlDataSource, the DataItem
            ' is generally returned as a DataRowView object
            Dim drv As System.Data.DataRowView = _
                CType(e.Row.DataItem, System.Data.DataRowView)

            If (drv("Region") Is System.DBNull.Value) Then
                e.Row.BackColor = System.Drawing.Color.Red
                e.Row.ForeColor = System.Drawing.Color.White
            End If
        End If
    End Sub
</script>
```

C#

```
<script runat="server">
    protected void GridView1_RowDataBound(object sender, GridViewRowEventArgs e)
    {
        //Test for null since gridview rows like the
        // Header and Footer will have a null DataItem
        if (e.Row.DataItem != null)
        {
            //Used to verify the DataItem object type
    System.Diagnostics.Debug.WriteLine(e.Row.DataItem.ToString());

            //When bound to a SqlDataSource, the DataItem
            // is generally returned as a DataRowView object
            System.Data.DataRowView drv = (System.Data.DataRowView)e.Row.DataItem;

            if (drv["Region"] == System.DBNull.Value)
            {
                e.Row.BackColor = System.Drawing.Color.Red;
                e.Row.ForeColor = System.Drawing.Color.White;
            }
        }
    }
</script>
```

The GridView also includes events that correspond to selecting, inserting, updating, and deleting data. You learn more about these events later in the chapter.

Using the EmptyDataText and EmptyDataTemplate Properties

In some cases, the data source bound to the GridView may not contain any data for the control to bind to. Even in these cases, you may want to provide the end user with some feedback, informing him that no data is present for the control to bind to. The GridView offers you two techniques to do this.

You first option is to use the `EmptyDataText` property. The property allows you to specify a string of text that is displayed to the user when no data is present for the GridView to bind to. When the ASP.NET page loads and the GridView determines there was no data available in its bound data source, it creates a special DataRow containing the `EmptyDataText` value and displays that to the user. Listing 7-24 shows how you can add this property to the GridView.

Listing 7-24: Adding EmptyDataText to the GridView

```
<asp:GridView ID="GridView1" Runat="server" DataSourceID="SqlDataSource1"
    DataKeyNames="CustomerID" AutoGenerateColumns="False"
    EmptyDataText="No data was found using your query">
```

The other option is to use an EmptyDataTemplate template to completely customize the special row the user sees when no data exists for the control to bind to.

> *A control template is simply a container that gives you the capability to add other content such as text, HTML controls, or even ASP.NET controls. The GridView control provides you with a variety of templates for various situations, including the EmptyDataTemplate template. You examine these templates throughout the rest of the GridView control section of this chapter.*

You can access the template from the Visual Studio 2008 design surface in two ways. The first option is to right-click the GridView control, expand the Edit Template option in the context menu, and select the EmptyDataTemplate item from the menu. This procedure is shown in Figure 7-14.

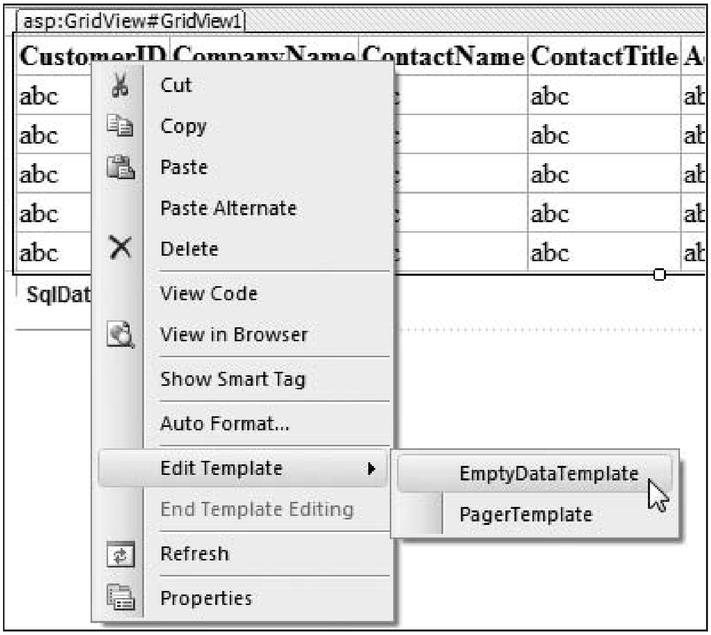

asp:GridView#GridView1
CustomerID
CompanyName
ContactName
ContactTitle
abc
SqlDat
Cut
Copy
Paste
Paste Alternate
Delete
View Code
View in Browser
Show Smart Tag
Auto Format...
Edit Template
End Template Editing
Refresh
Properties
EmptyDataTemplate
PagerTemplate

Figure 7-14

The other option for opening the template is to select the Edit Templates option from the GridView smart tag. Selecting this option puts the GridView into template editing mode and presents you with a dialog from which you can choose the specific template you wish to edit. Simply select EmptyDataTemplate from the drop-down list, as shown in Figure 7-15.

After you have entered template editing mode, you can simply add your custom text and/or controls to the template editor on the design surface. When you have finished editing the template, simply right-click, or open the GridViews smart tag and select the End Template Editing option.

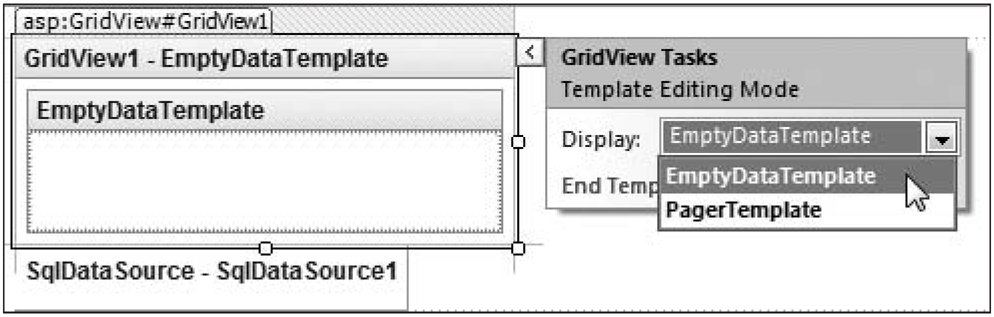

asp:GridView#GridView1
GridView1 - EmptyDataTemplate
EmptyDataTemplate
SqlDataSource - SqlDataSource1
GridView Tasks
Template Editing Mode
Display:
EmptyDataTemplate
End Temp
EmptyDataTemplate
PagerTemplate

Figure 7-15

Switching to Source view, you see that an `<EmptyDataTemplate>` element has been added to the GridView control. The element should contain all the controls and HTML that you added while editing the template. Listing 7-25 shows an example of an `EmptyDataTemplate`.

Listing 7-25: Using EmptyDataTemplate

```
<EmptyDataTemplate>
    <table style="width: 225px">
        <tr>
            <td colspan="2">
                No data could be found based on your query parameters.
                Please enter a new query.</td>
        </tr>
        <tr>
            <td style="width: 162px">
                <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox></td>
            <td style="width: 102px">
                <asp:Button ID="Button1" runat="server" Text="Search" /></td>
        </tr>
    </table>
</EmptyDataTemplate>
```

You could, of course, have also added the template and its contents while in Source view.

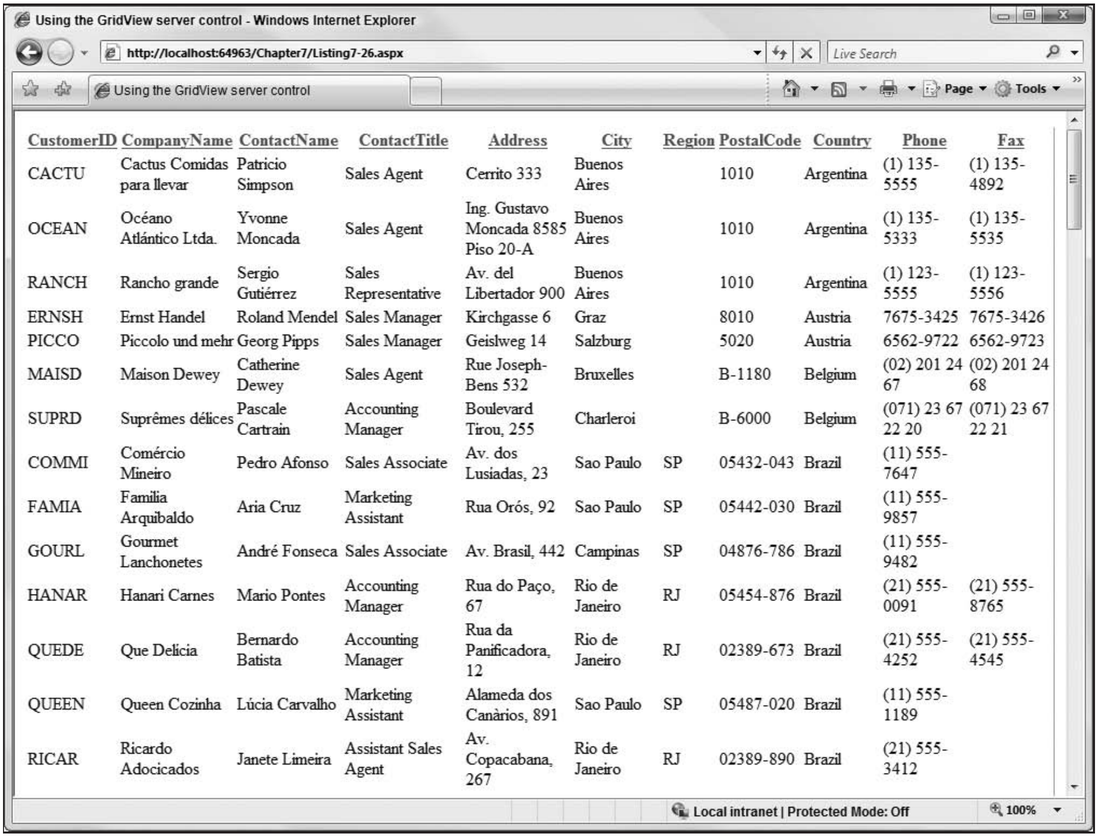
Enabling GridView Column Sorting

The capability to sort data is one of the most basic tools users have to navigate through a significant amount of data. The DataGrid control made sorting columns in a grid a relatively easy task, but the GridView control takes it one step further. Unlike using the DataGrid, where you are responsible for coding the sort routine, to enable column sorting in this grid, you just set the `AllowSorting` attribute to `True`. The control takes care of all the sorting logic for you internally. Listing 7-26 shows how to add this attribute to your grid.

Listing 7-26: Adding sorting to the GridView control

```
<asp:GridView ID="GridView1" Runat="server" DataSourceID="SqlDataSource1"
     DataKeyNames="CustomerID" AutoGenerateColumns="False"
     AllowSorting="True">
```

After enabling sorting, you see that all grid columns have now become hyperlinks. Clicking a column header sorts that specific column. Figure 7-16 shows your grid after the data has been sorted by country.


Using the GridView server control - Windows Internet Explorer
http://localhost:64963/Chapter7/Listing7-26.aspx
Live Search
Using the GridView server control
Page
Tools
Local intranet | Protected Mode: Off
100%

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
CACTU	Cactus Comidas para llevar	Patricio Simpson	Sales Agent	Cerrito 333	Buenos Aires		1010	Argentina	(1) 135-5555	(1) 135-4892
OCEAN	Océano Atlántico Ltda.	Yvonne Moncada	Sales Agent	Ing. Gustavo Moncada 8585 Piso 20-A	Buenos Aires		1010	Argentina	(1) 135-5333	(1) 135-5535
RANCH	Rancho grande	Sergio Gutiérrez	Sales Representative	Av. del Libertador 900	Buenos Aires		1010	Argentina	(1) 123-5555	(1) 123-5556
ERNSH	Ernst Handel	Roland Mendel	Sales Manager	Kirchgasse 6	Graz		8010	Austria	7675-3425	7675-3426
PICCO	Piccolo und mehr	Georg Pipps	Sales Manager	Geislweg 14	Salzburg		5020	Austria	6562-9722	6562-9723
MAISD	Maison Dewey	Catherine Dewey	Sales Agent	Rue Joseph-Bens 532	Bruxelles		B-1180	Belgium	(02) 201 24 67	(02) 201 24 68
SUPRD	Suprêmes délices	Pascale Cartrain	Accounting Manager	Boulevard Tirou, 255	Charleroi		B-6000	Belgium	(071) 23 67 22 20	(071) 23 67 22 21
COMMI	Comércio Mineiro	Pedro Afonso	Sales Associate	Av. dos Lusiadas, 23	Sao Paulo	SP	05432-043	Brazil	(11) 555-7647	
FAMIA	Familia Arquibaldo	Aria Cruz	Marketing Assistant	Rua Orós, 92	Sao Paulo	SP	05442-030	Brazil	(11) 555-9857	
GOURL	Gourmet Lanchonetes	André Fonseca	Sales Associate	Av. Brasil, 442	Campinas	SP	04876-786	Brazil	(11) 555-9482	
HANAR	Hanari Carnes	Mario Pontes	Accounting Manager	Rua do Paço, 67	Rio de Janeiro	RJ	05454-876	Brazil	(21) 555-0091	(21) 555-8765
QUEDE	Que Delícia	Bernardo Batista	Accounting Manager	Rua da Panificadora, 12	Rio de Janeiro	RJ	02389-673	Brazil	(21) 555-4252	(21) 555-4545
QUEEN	Queen Cozinha	Lúcia Carvalho	Marketing Assistant	Alameda dos Canàrios, 891	Sao Paulo	SP	05487-020	Brazil	(11) 555-1189	
RICAR	Ricardo Adocicados	Janete Limeira	Assistant Sales Agent	Av. Copacabana, 267	Rio de Janeiro	RJ	02389-890	Brazil	(21) 555-3412	

Figure 7-16

GridView sorting has also been enhanced in a number of other ways. The grid can handle both ascending and descending sorting. If you repeatedly click on a column header, you cause the sort order to switch back and forth between ascending and descending. The GridView's `Sort` method can also accept multiple `SortExpressions` to enable multicolumn sorting. Listing 7-27 shows how you can use the GridView's sorting event to implement a multicolumn sort.

Listing 7-27: Adding multicolumn sorting to the GridView

VB

```
<script runat="server">
    Protected Sub GridView1_Sorting(ByVal sender As Object, _
       ByVal e As GridViewSortEventArgs)

        Dim oldExpression As String = GridView1.SortExpression
        Dim newExpression As String = e.SortExpression
        If (oldExpression.IndexOf(newExpression) < 0) Then
            If (oldExpression.Length > 0) Then
                e.SortExpression = newExpression & "," & oldExpression
            Else
                e.SortExpression = newExpression
            End If
        Else
            e.SortExpression = oldExpression
        End If
    End Sub
</script>
```

C#

```
<script runat="server">
    protected void GridView1_Sorting(object sender, GridViewSortEventArgs e)
    {
        string oldExpression = GridView1.SortExpression;
        string newExpression = e.SortExpression;

        if (oldExpression.IndexOf(newExpression) < 0)
        {
            if (oldExpression.Length > 0)
                e.SortExpression = newExpression + "," + oldExpression;
            else
                e.SortExpression = newExpression;
        }
        else
        {
            e.SortExpression = oldExpression;
        }
    }
</script>
```

Notice the listing uses the GridView's `Sorting` event to manipulate the value of the control's `SortExpression` property. The events parameters enable you to examine the current sort expression, direct the sort, or even cancel the sort action altogether. The GridView also offers a `Sorted` event which is raised after the sort has completed.

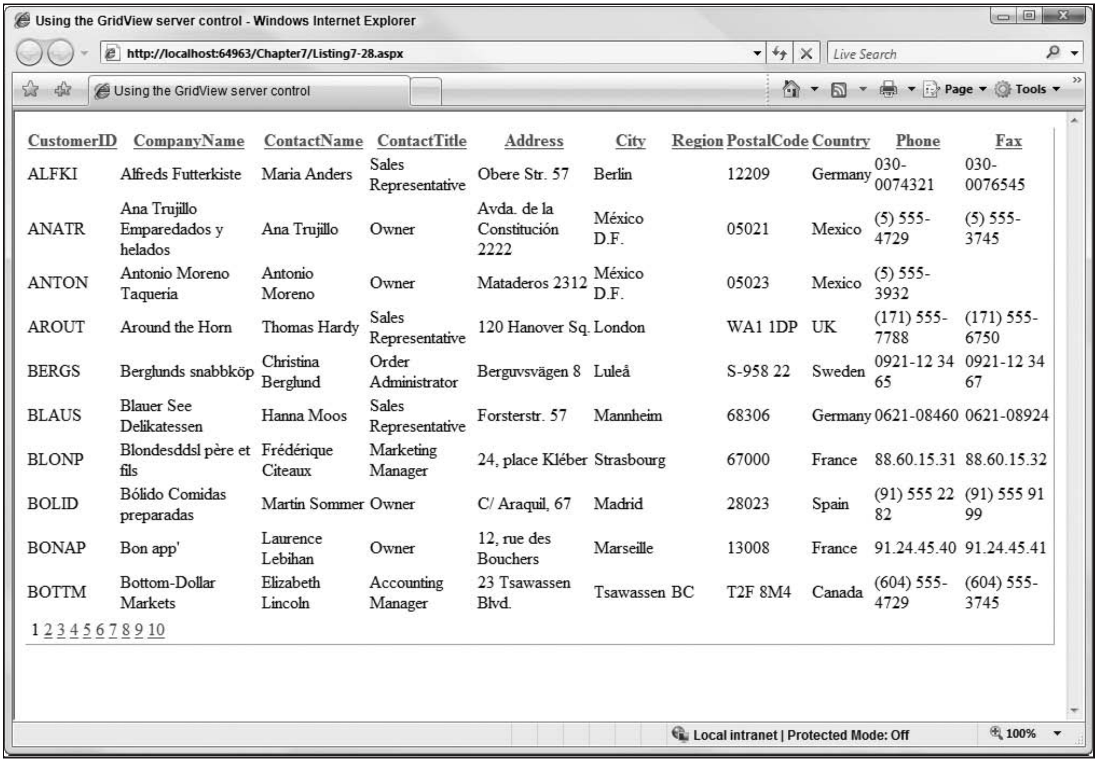
Enabling the GridView Pager

The GridView also greatly improves upon another common grid navigation feature — paging. Although the implementation of paging using a DataGrid greatly simplified paging (especially in comparison to paging in ASP), the GridView makes it even easier with its `AllowPaging` property. This property can be set either by adding the attribute to the GridView control in HTML mode or by checking the Enable Paging check box in the GridView's smart tag. Enabling paging in the GridView control defaults to a page size of 10 records and adds the Pager to the bottom of the grid. Listing 7-28 shows an example of modifying your grid to enable sorting and paging.

Listing 7-28: Enabling sorting and paging on the GridView control

```
<asp:GridView ID="GridView1" Runat="server" DataSourceID="SqlDataSource1"
     DataKeyNames="CustomerID" AutoGenerateColumns="False"
      AllowSorting="True" AllowPaging="True">
```

Enabling sorting and paging in your GridView creates a page that looks like Figure 7-17.


Using the GridView server control - Windows Internet Explorer
http://localhost:64963/Chapter7/Listing7-28.aspx
Live Search
Using the GridView server control
Page
Tools
Local intranet | Protected Mode: Off
100%

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076545
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK	(171) 555-7788	(171) 555-6750
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65	0921-12 34 67
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany	0621-08460	0621-08924
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France	88.60.15.31	88.60.15.32
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67	Madrid		28023	Spain	(91) 555 22 82	(91) 555 91 99
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille		13008	France	91.24.45.40	91.24.45.41
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen	BC	T2F 8M4	Canada	(604) 555-4729	(604) 555-3745

1 2 3 4 5 6 7 8 9 10

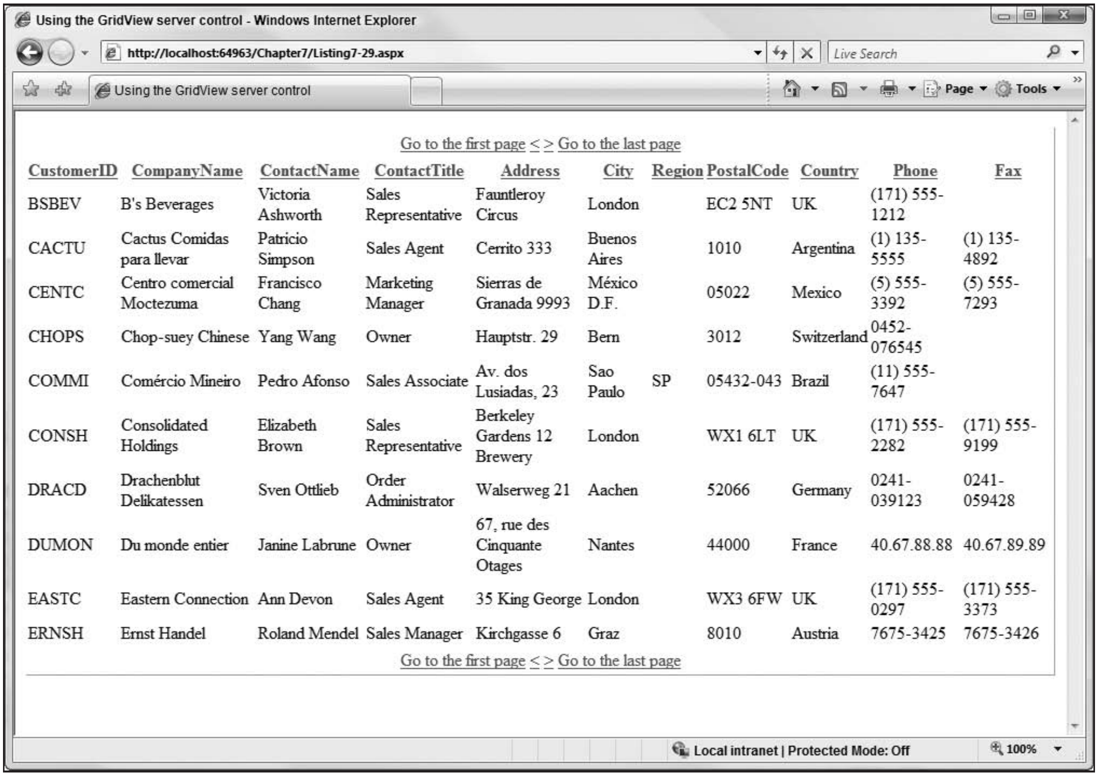
Figure 7-17

As with the DataGrid, the GridView allows most of the paging options to be customized. For instance, the `PagersSettings-Mode` attribute allows you to dictate how the grid's Pager is displayed using the various Pager modes including `NextPrevious`, `NextPreviousFirstLast`, `Numeric` (the default value), or `NumericFirstLast`. Additionally, by specifying the `PagerStyle` element in the GridView, you can customize how the grid displays the Pager text, including font color, size, and type, as well as text alignment and a variety of other style options. Listing 7-29 shows how you might customize your GridView control to use the `NextPrevious` mode and style the Pager text using the `PagerStyle` element. Also, you can control the number of records displayed on the page using the GridView's `PageSize` attribute.

Listing 7-29: Using the PagerStyle and PagerSettings properties in the GridView control

```
<asp:GridView ID="GridView1" Runat="server" DataSourceID="SqlDataSource1"
    DataKeyNames="CustomerID" AutoGenerateColumns="False"
     AllowSorting="True" AllowPaging="True" PageSize="10">
     <PagerStyle HorizontalAlign="Center"></PagerStyle>
     <PagerSettings Position="TopAndBottom"
         FirstPageText="Go to the first page"
         LastPageText="Go to the last page" Mode="NextPreviousFirstLast">
     </PagerSettings>
```

Figure 7-18 shows the grid after you change several style options and set the `PagerSettings-Mode` to `NextPreviousFirstLast`.


Using the GridView server control - Windows Internet Explorer
http://localhost:64963/Chapter7/Listing7-29.aspx
Live Search
Using the GridView server control
Page
Tools
Local intranet | Protected Mode: Off
100%

Go to the first page < > Go to the last page

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative	Fauntleroy Circus	London		EC2 5NT	UK	(171) 555-1212	
CACTU	Cactus Comidas para llevar	Patricio Simpson	Sales Agent	Cerrito 333	Buenos Aires		1010	Argentina	(1) 135-5555	(1) 135-4892
CENTC	Centro comercial Moctezuma	Francisco Chang	Marketing Manager	Sierras de Granada 9993	México D.F.		05022	Mexico	(5) 555-3392	(5) 555-7293
CHOPS	Chop-suey Chinese	Yang Wang	Owner	Hauptstr. 29	Bern		3012	Switzerland	0452-076545	
COMMI	Comércio Mineiro	Pedro Afonso	Sales Associate	Av. dos Lusiadas, 23	Sao Paulo	SP	05432-043	Brazil	(11) 555-7647	
CONSH	Consolidated Holdings	Elizabeth Brown	Sales Representative	Berkeley Gardens 12 Brewery	London		WX1 6LT	UK	(171) 555-2282	(171) 555-9199
DRACD	Drachenblut Delikatessen	Sven Ottlieb	Order Administrator	Walserweg 21	Aachen		52066	Germany	0241-039123	0241-059428
DUMON	Du monde entier	Janine Labrune	Owner	67, rue des Cinquante Otages	Nantes		44000	France	40.67.88.88	40.67.89.89
EASTC	Eastern Connection	Ann Devon	Sales Agent	35 King George	London		WX3 6FW	UK	(171) 555-0297	(171) 555-3373
ERNSH	Ernst Handel	Roland Mendel	Sales Manager	Kirchgasse 6	Graz		8010	Austria	7675-3425	7675-3426

Go to the first page < > Go to the last page

Figure 7-18

The GridView has a multitude of other Pager and Pager style options that we encourage you to experiment with. Because the list of `PagerSetting` and `PagerStyle` options is so long, all options are not listed here. You can find a full list of the options in the Visual Studio Help documents.

Additionally, the GridView control offers two events you can use to alter the standard paging behavior of the grid. The `PageIndexChanging` and `PageIndexChanged`events are raised before and after the GridView's current page index changes. The page index changes when the user clicks the pager links in the grid. The `PageIndexChanging` event parameters allow you to examine the value of the new page index before it actually changes or even cancel the Paging event altogether.

The GridView also includes the `EnableSortingAndPagingCallbacks` property that allows you to indicate whether the control should use client callbacks to perform sorting and paging. Client callbacks can help your user avoid suffering through a complete page postback for operations such as sorting and paging the GridView. Instead of requiring a complete page postback, client callbacks use AJAX to perform the sort and page actions.

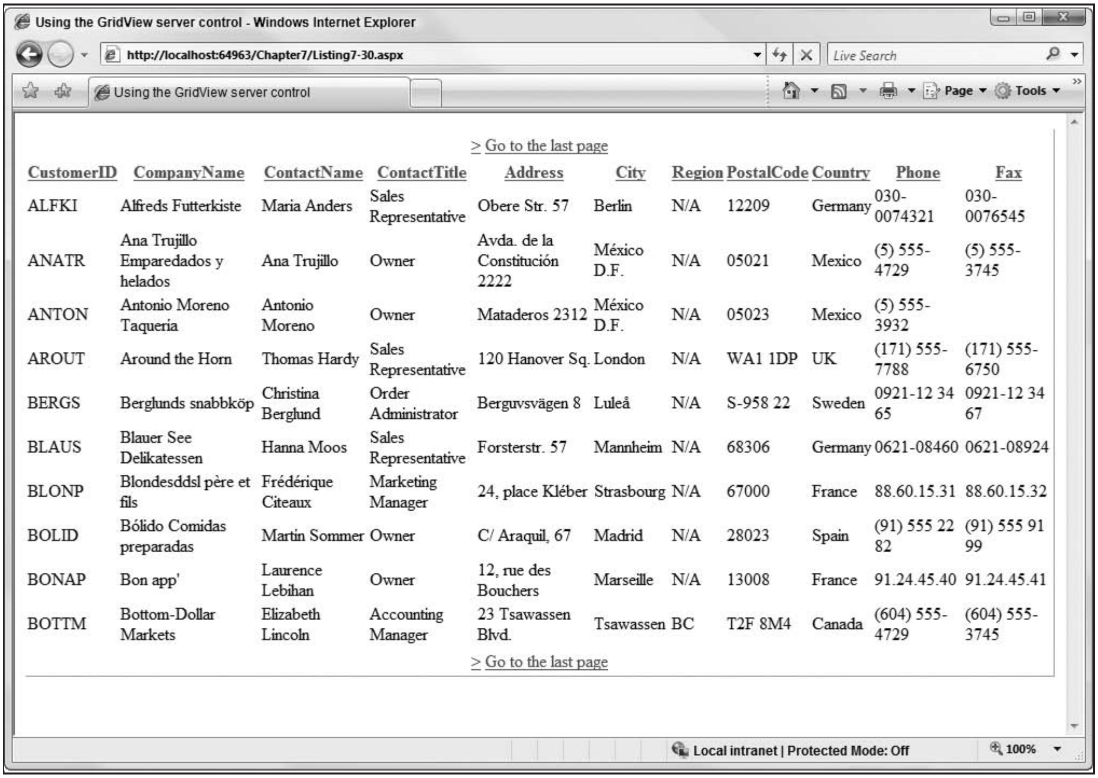
If you are interested in learning more about other ways you can integrate AJAX into your ASP.NET applications, Chapters 19 and 20 introduce you to the ASP.NET AJAX framework and how you can leverage its capabilities in your applications.

Another interesting feature of column generation is the capability to specify what the GridView should display when it encounters a `Null` value within the column. For an example of this, add a column using an additional `<asp:BoundField>` control, as shown in Listing 7-30.

Listing 7-30: Using the Null value

```
<asp:BoundField HeaderText="Region" NullDisplayText="N/A"
    DataField="Region" SortExpression="Region"></asp:BoundField>
```

In this example, the `<asp:BoundField>` element displays the Region column from the Customers table. As you look through the data in the Region section, notice that not every row has a value in it. If you don't want to display just a blank box to show an empty value, you can use some text in place of the empty items in the column. For this, you utilize the `NullDisplayText` attribute. The `String` value it provides is used for each and every row that doesn't have a Region value. This construct produces the results illustrated in Figure 7-19.


Using the GridView server control - Windows Internet Explorer
http://localhost:64963/Chapter7/Listing7-30.aspx
Live Search
Using the GridView server control
Page
Tools
Local intranet | Protected Mode: Off
100%

> Go to the last page

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin	N/A	12209	Germany	030-0074321	030-0076545
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.	N/A	05021	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.	N/A	05023	Mexico	(5) 555-3932	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London	N/A	WA1 1DP	UK	(171) 555-7788	(171) 555-6750
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå	N/A	S-958 22	Sweden	0921-12 34 65	0921-12 34 67
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim	N/A	68306	Germany	0621-08460	0621-08924
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg	N/A	67000	France	88.60.15.31	88.60.15.32
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67	Madrid	N/A	28023	Spain	(91) 555 22 82	(91) 555 91 99
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille	N/A	13008	France	91.24.45.40	91.24.45.41
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen	BC	T2F 8M4	Canada	(604) 555-4729	(604) 555-3745

> Go to the last page

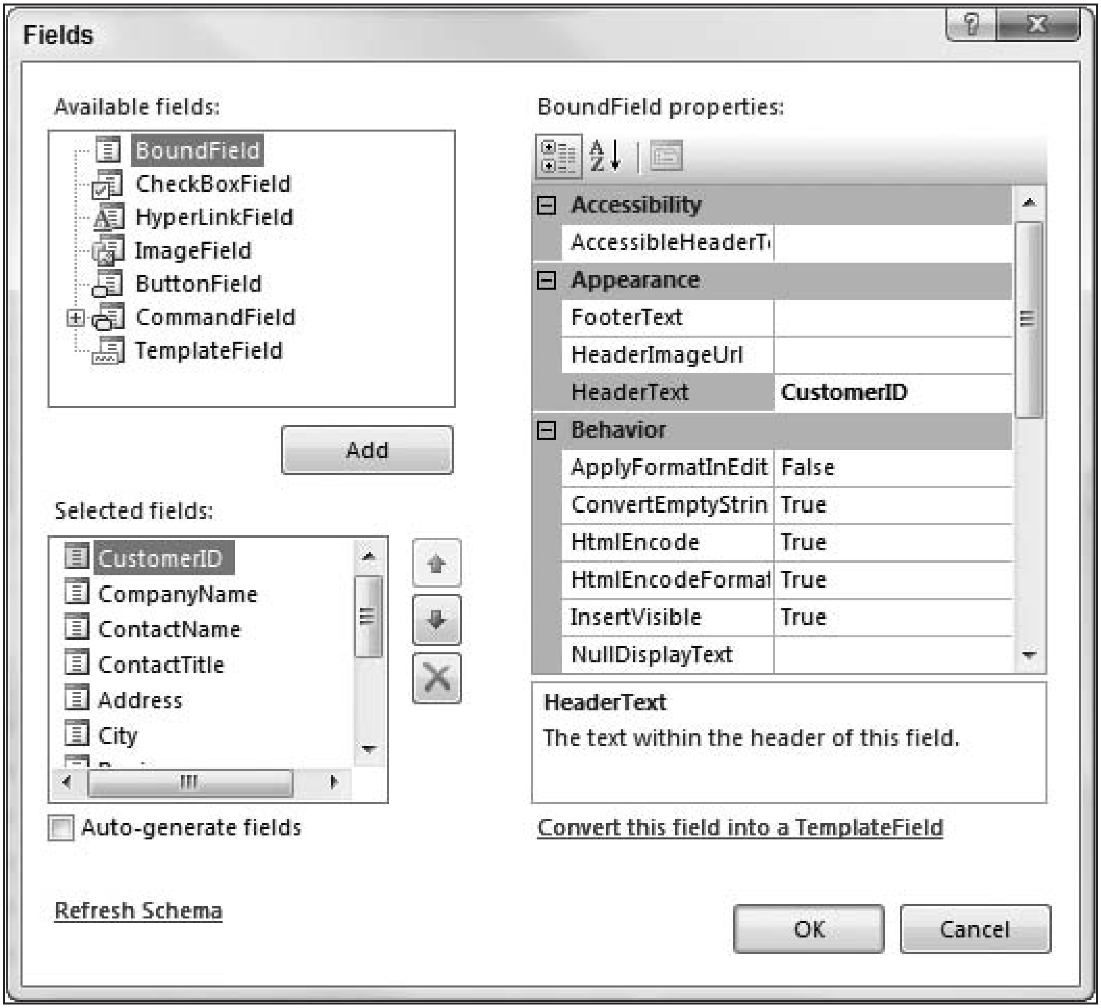
Figure 7-19

Customizing Columns in the GridView

Frequently, the data in your grid is not simply text data, but data that you either want to display using other types of controls or don't want to display at all. For instance, you have been retrieving the CustomerID as part of your `SELECT` query and displaying it in your grid. By default, the GridView control displays all columns returned as part of a query. But rather than automatically displaying the CustomerID, it might be better to hide that data from the end user. Or perhaps you are also storing the corporate URL for all your customers and want the CustomerName column to link directly to their Web sites. The GridView gives you great flexibility and power regarding how you display the data in your grid.

> *The GridView automatically creates a* `CheckBoxField` *for columns with a data type of bit or Boolean*

You can edit your GridView columns in two ways. You can select the Edit Columns option from the GridView smart tag. This link allows you to edit any existing columns in your grid using the Fields dialog window, shown in Figure 7-20. From here you can change a column's visibility, header text, the usual style options, and many other properties of the column.


Fields
Available fields:
BoundField
CheckBoxField
HyperLinkField
ImageField
ButtonField
CommandField
TemplateField
Add
Selected fields:
CustomerID
CompanyName
ContactName
ContactTitle
Address
City
BoundField properties:
Accessibility
Appearance
Behavior
Convert this field into a TemplateField
Refresh Schema
OK
Cancel

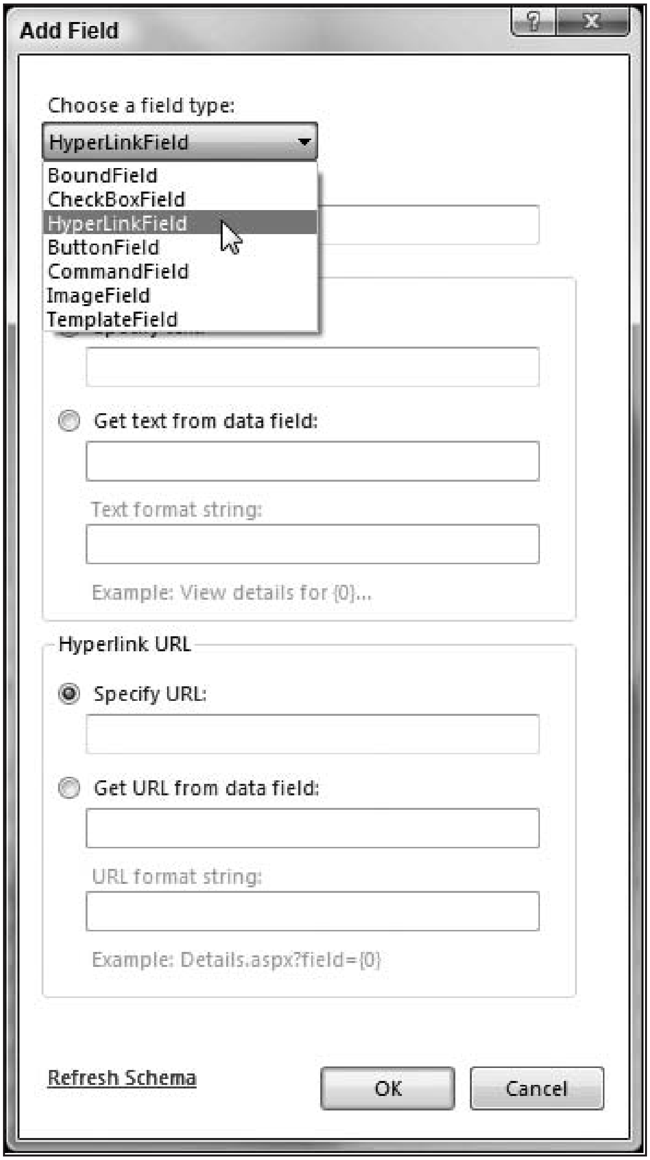
AccessibleHeaderT	
FooterText	
HeaderImageUrl	
HeaderText	CustomerID
ApplyFormatInEdit	False
ConvertEmptyStrin	True
HtmlEncode	True
HtmlEncodeFormat	True
InsertVisible	True
NullDisplayText	

HeaderText
The text within the header of this field.

☐ Auto-generate fields

Figure 7-20

Selecting the Add New Column link from the GridView control's smart tag displays another easy form — the Add Field dialog (shown in Figure 7-21) — with options allowing you to add completely new columns to your grid. Depending on which column field type you select from the drop-down list, the dialog presents you with the appropriate options for that column type. In this case, you want to add a hyperlink; so you select the HyperLinkField from the drop-down list. The Add Field dialog changes and lets you enter in the hyperlink information, including the URL, the data field, and a format string for the column.


Choose a field type: HyperLinkField

Hyperlink URL: ● Specify URL:

Add Field
BoundField
CheckBoxField
HyperLinkField
ButtonField
CommandField
ImageField
TemplateField
○ Get text from data field:
Text format string:
Example: View details for {0}...
○ Get URL from data field:
URL format string:
Example: Details.aspx?field={0}
Refresh Schema
OK
Cancel

Figure 7-21

The Add Field dialog lets you select one of the field types described in the following table.

Field Control	Description
BoundField	Displays the value of a field in a data source. This is the default column type of the GridView control.
CheckBoxField	Displays a check box for each item in the GridView control. This column field type is commonly used to display fields with a Boolean value.
HyperLinkField	Displays the value of a field in a data source as a hyperlink. This column field type allows you to bind a second field to the hyperlink's URL.
ButtonField	Displays a command button for each item in the GridView control. This allows you to create a column of custom button controls, such as an Add or Remove button.
CommandField	Represents a field that displays command buttons to perform select, edit, insert, or delete operations in a data-bound control.
ImageField	Automatically displays an image when the data in the field represents an image.
TemplateField	Displays user-defined content for each item in the GridView control according to a specified template. This column field type allows you to create a customized column field.

You can also change the grid columns in the Source view. Listing 7-31 shows how you can add a HyperLinkField. Note that by providing a comma-delimited list of data field names, you can specify multiple data fields to bind to this column. You can then use these fields in your format string to pass two query string parameters.

Listing 7-31: Adding a HyperLinkField control to the GridView

```
<asp:HyperLinkField HeaderText="CompanyName"
    DataNavigateUrlFields="CustomerID,Country" SortExpression="CompanyName"
    DataNavigateUrlFormatString=
        "http://www.foo.com/Customer.aspx?id={0}&country={1}"
    DataTextField="CompanyName">
</asp:HyperLinkField>
```

Using the TemplateField Column

A key column type available in the GridView control is the TemplateField column that enables you to use templates to completely customize the contents of the column.

> *As described earlier in the GridView section of this chapter, a control template is simply a container to which you can add other content, such as text, HTML controls, or even ASP.NET controls.*

The TemplateField provides you with six different templates that enable you to customize a specific area of the column or to create a mode that a cell in the column may enter, such as edit mode. The following table describes the available templates.

Template Name	Description
ItemTemplate	Template used for displaying an item in the TemplateField of the data-bound control
AlternatingItem-Template	Template used for displaying the alternating items of the TemplateField
EditItemTemplate	Template used for displaying a TemplateField item in edit mode
InsertItemTemplate	Template used for displaying a TemplateField item in insert mode
HeaderTemplate	Template used for displaying the header section of the TemplateField
FooterTemplate	Template used for displaying the footer section of the TemplateField

To use the TemplateField in a GridView, simply add the column type to your grid using the Add Field dialog as described in the previous section. After you have added the field, note that a new `<asp:TemplateField>` tag (like the one shown in Listing 7-32) has been added to the GridView.

Listing 7-32: The GridViews TemplateField

```
<asp:TemplateField></asp:TemplateField>
```

This element serves as the container for the various templates the column can contain. Now that you have added the column, you create your custom content. You can do this by using the template editing features of the Visual Studio 2008 design surface or by adding your own content to the TemplateField element in Source view.

You can access the template editing features from the Visual Studio design surface in two ways. The first method is to right-click the GridView and choose the Column[nn] (where *nn* is your specific column index) option from the Edit Template option in the context menu. When you use this method, each available template for the column is displayed on the Visual Studio 2008 design surface, as shown in Figure 7-22.

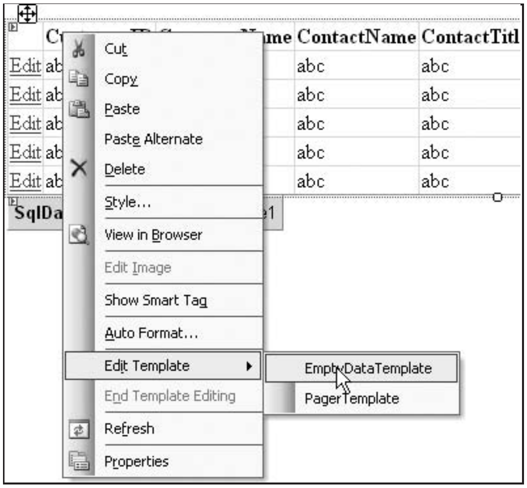

ContactName
ContactTitl
Edit
abc
SqlDa
Cut
Copy
Paste
Paste Alternate
Delete
Style...
View in Browser
Edit Image
Show Smart Tag
Auto Format...
Edit Template
End Template Editing
Refresh
Properties
EmptyDataTemplate
PagerTemplate

Figure 7-22

The second option is to open the GridView control's smart tag and select the Edit Template command. This opens a menu, shown in Figure 7-23, which allows you to select the column template you want to edit.

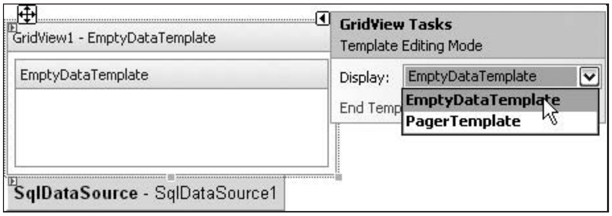

GridView1 - EmptyDataTemplate
EmptyDataTemplate
GridView Tasks
Template Editing Mode
Display:
EmptyDataTemplate
End Temp
EmptyDataTemplate
PagerTemplate
SqlDataSource - SqlDataSource1

Figure 7-23

The ItemTemplate is probably the template you will use the most because it controls the default contents of each cell of the column. Listing 7-33 demonstrates how you can use the ItemTemplate to customize the contents of the column.

Listing 7-33: Using ItemTemplate

```
<asp:TemplateField HeaderText="CurrentStatus">
    <ItemTemplate>
        <table>
            <tr>
                <td align="center" style="width: 78px">
                    <asp:Button ID="Button2" runat="server" Text="Enable" /></td>
                <td align="center" style="width: 76px">
                    <asp:Button ID="Button3" runat="server" Text="Disable" /></td>
            </tr>
        </table>
    </ItemTemplate>
</asp:TemplateField>
```

Notice that, in the sample, the ItemTemplate contains a combination of HTML and ASP.NET controls. Figure 7-24 shows what the sample looks like when it is displayed in the browser.

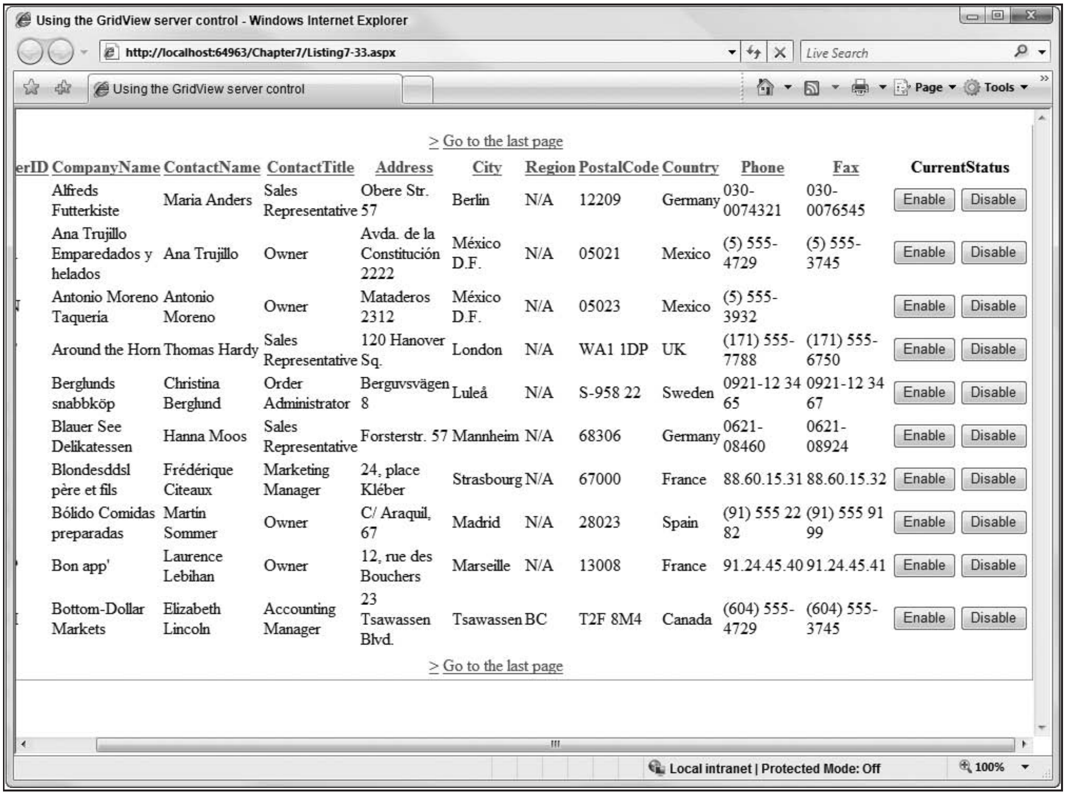
Because the GridView control is data-bound, you can also access the data being bound to the control using data-binding expressions such as the Eval, XPath, or Bind expressions. For instance, Listing 7-34 shows how you can add a data-binding expression using the `Eval` method to set the text field of the Button control. You can read more about data-binding expressions later in this chapter.

Listing 7-34: Adding a data-binding expression

```
<asp:TemplateField HeaderText="CurrentStatus">
    <ItemTemplate>
        <table>
            <tr>
                <td align="center" style="width: 78px">
                    <asp:Button ID="Button2" runat="server"
                        Text='<%# "Enable " + Eval("CustomerID") %>' /></td>
                <td align="center" style="width: 76px">
```

Continued

```
                    <asp:Button ID="Button3" runat="server"
                        Text='<%# "Disable " + Eval("CustomerID") %>' />
                </td>
            </tr>
        </table>
    </ItemTemplate>
</asp:TemplateField>
```


Using the GridView server control - Windows Internet Explorer
http://localhost:64963/Chapter7/Listing7-33.aspx
Live Search
Using the GridView server control
Page
Tools
Local intranet | Protected Mode: Off
100%

> Go to the last page

erID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax	CurrentStatus
	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin	N/A	12209	Germany	030-0074321	030-0076545	Enable Disable
	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.	N/A	05021	Mexico	(5) 555-4729	(5) 555-3745	Enable Disable
	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.	N/A	05023	Mexico	(5) 555-3932		Enable Disable
	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London	N/A	WA1 1DP	UK	(171) 555-7788	(171) 555-6750	Enable Disable
	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå	N/A	S-958 22	Sweden	0921-12 34 65	0921-12 34 67	Enable Disable
	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim	N/A	68306	Germany	0621-08460	0621-08924	Enable Disable
	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg	N/A	67000	France	88.60.15.31	88.60.15.32	Enable Disable
	Bólido Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67	Madrid	N/A	28023	Spain	(91) 555 22 82	(91) 555 91 99	Enable Disable
	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille	N/A	13008	France	91.24.45.40	91.24.45.41	Enable Disable
	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen	BC	T2F 8M4	Canada	(604) 555-4729	(604) 555-3745	Enable Disable

> Go to the last page

Figure 7-24

Other common templates are the InsertTemplate and EditTemplate. These templates are used by the grid when a grid row enters insert or edit mode. You examine inserting and editing data in the GrivView control, including using the InsertItemTemplate and EditItemTemplate, in the next section.

Editing GridView Row Data

Not only do users want to view the data in their browser, but they also want to be able to edit the data and save changes back to the data store. Adding editing capabilities to the DataGrid was never easy, but it was important enough that developers frequently attempted to do so.

The GridView control makes it very easy to edit the data contained in the grid. To demonstrate just how easy it is, you can modify the existing grid so you can edit the customer data it contains. First, modify your existing SqlDataSource control by adding an `UpdateCommand` attribute. This tells the data source control what SQL it should execute when it is requested to perform an update. Listing 7-35 shows the code to add the `UpdateCommand` attribute.

Listing 7-35: Adding an UpdateCommand to a SqlDataSource control

```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server"
    SelectCommand="SELECT * FROM [Customers]"
    ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>"
    DataSourceMode="DataSet"
    UpdateCommand="UPDATE [Customers] SET [CompanyName] = @CompanyName,
        [ContactName] = @ContactName, [ContactTitle] = @ContactTitle,
        [Address] = @Address, [City] = @City, [Region] = @Region,
        [PostalCode] = @PostalCode, [Country] = @Country, [Phone] = @Phone,
        [Fax] = @Fax WHERE [CustomerID] = @original_CustomerID">
```

Notice that the UpdateCommand includes a number of placeholders such as `@CompanyName`, `@Country`, `@Region`, and `@CustomerID`. These are placeholders for the corresponding information that will come from the selected row in GridView. In order to use the parameters, you must define them using the `UpdateParameters` element of the SqlDataSource control. The `UpdateParameters` element, shown in Listing 7-36, works much like the `SelectParameters` element discussed earlier in the chapter.

Listing 7-36: Adding UpdateParameters to the SqlDataSource control

```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server"
     SelectCommand="SELECT * FROM [Customers]"
     ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>"
     DataSourceMode="DataSet"
     UpdateCommand="UPDATE [Customers] SET [CompanyName] = @CompanyName,
          [ContactName] = @ContactName, [ContactTitle] = @ContactTitle,
          [Address] = @Address, [City] = @City, [Region] = @Region,
          [PostalCode] = @PostalCode, [Country] = @Country, [Phone] = @Phone,
          [Fax] = @Fax WHERE [CustomerID] = @original_CustomerID">
      <UpdateParameters>
          <asp:Parameter Type="String" Name="CompanyName"></asp:Parameter>
          <asp:Parameter Type="String" Name="ContactName"></asp:Parameter>
          <asp:Parameter Type="String" Name="ContactTitle"></asp:Parameter>
          <asp:Parameter Type="String" Name="Address"></asp:Parameter>
          <asp:Parameter Type="String" Name="City"></asp:Parameter>
          <asp:Parameter Type="String" Name="Region"></asp:Parameter>
          <asp:Parameter Type="String" Name="PostalCode"></asp:Parameter>
          <asp:Parameter Type="String" Name="Country"></asp:Parameter>
          <asp:Parameter Type="String" Name="Phone"></asp:Parameter>
          <asp:Parameter Type="String" Name="Fax"></asp:Parameter>
          <asp:Parameter Type="String" Name="CustomerID"></asp:Parameter>
      </UpdateParameters>
 </asp:SqlDataSource>
```

Within the `UpdateParameters` element, each named parameter is defined using the `<asp:Parameter>` element. This element uses two attributes that define the name and the data type of the parameter. In this case, all the parameters are of type `String`. Remember that you can also use any of the parameter types mentioned earlier in the chapter, such as the `ControlParameter` or `QueryStringParameter` in the `UpdateParameters` element.

Next, you give the grid a column it can use to trigger editing of a data row. You can do this in several ways. First, you can use the GridView's `AutoGenerateEditButton` property. When set to `True`, this property tells the grid to add to itself a `ButtonField` column with an edit button for each data row. Listing 7-37 shows how to add the `AutoGenerateEditButton` attribute to the GridView control.

Listing 7-37: Adding the AutoGenerateEditButton attribute to a SqlDataSource control

```
<asp:GridView ID="GridView1" Runat="server" DataSourceID="SqlDataSource1"
     DataKeyNames="CustomerID" AutoGenerateColumns="False"
     AllowSorting="True" AllowPaging="True"
     AutoGenerateEditButton="true">
```

The GridView control also includes `AutoGenerateSelectButton` and `AutoGenerateDeleteButton` properties, which allow you to easily add row selection and row deletion capabilities to the grid.

A second way to add an edit button is to add a `CommandField` column. This is shown in Listing 7-38.

Listing 7-38: Adding edit functionality using a CommandField

```
<asp:CommandField ShowHeader="True" HeaderText="Command"
     ShowEditButton="True"></asp:CommandField>
```

Notice that you add the `ShowEditButton` property to the `CommandField` to indicate that you want to display the edit command in this column. You can control how the command is displayed in the grid using the `ButtonType` property, which allows you to display the command as a link, a button, or even an image. Figure 7-25 shows what the grid looks like after adding the `CommandField` with the `edit` command displayed.

Now if you browse to your Web page, you see that a new edit column has been added. Clicking the Edit link allows the user to edit the contents of that particular data row.

The `CommandField` element also has attributes that allow you to control exactly what is shown in the column. You can dictate whether the column displays commands such as `Cancel`, `Delete`, `Edit`, `Insert`, and `Select`.

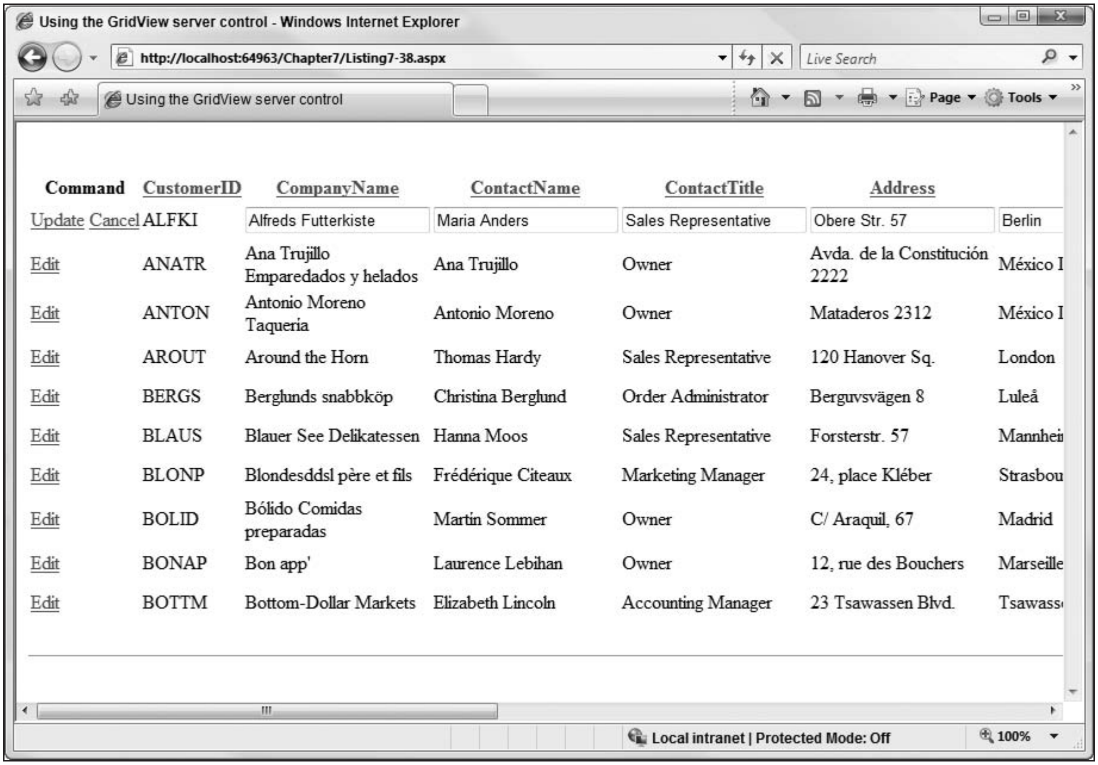
With the Edit `CommandField` enabled, you still have one more property to set in order to enable the grid to perform updates. You tell the grid which columns are in the table's primary key. You can accomplish this by using the `DataKeyNames` property, as illustrated in Listing 7-39.

Listing 7-39: Adding the DataKeyNames to the GridView control

```
<asp:GridView ID="GridView1" Runat="server" DataSourceID="SqlDataSource1"
      DataKeyNames="CustomerID" AutoGenerateColumns="False"
     AllowSorting="True" AllowPaging="True"
     AutoGenerateEditButton="true">
```

You can specify more than one column in the primary key by setting the property to a comma-delimited list.

Notice that when you add the edit capabilities to the grid, by default it allows all displayed columns to be edited. You probably won't always want this to be the case. You can control which columns the grid allows to be edited by adding the `ReadOnly` property to the columns that you do not want users to edit. Listing 7-40 shows how you can add the `ReadOnly` property to the ID column.


Using the GridView server control - Windows Internet Explorer
http://localhost:64963/Chapter7/Listing7-38.aspx
Live Search
Using the GridView server control
Page
Tools
Local intranet | Protected Mode: Off
100%

Command	CustomerID	CompanyName	ContactName	ContactTitle	Address	
Update Cancel	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin
Edit	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México I
Edit	ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	México I
Edit	AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London
Edit	BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå
Edit	BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannhei
Edit	BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbou
Edit	BOLID	Bólido Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67	Madrid
Edit	BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille
Edit	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawass

Figure 7-25

Listing 7-40: Adding the ReadOnly property to a BoundField

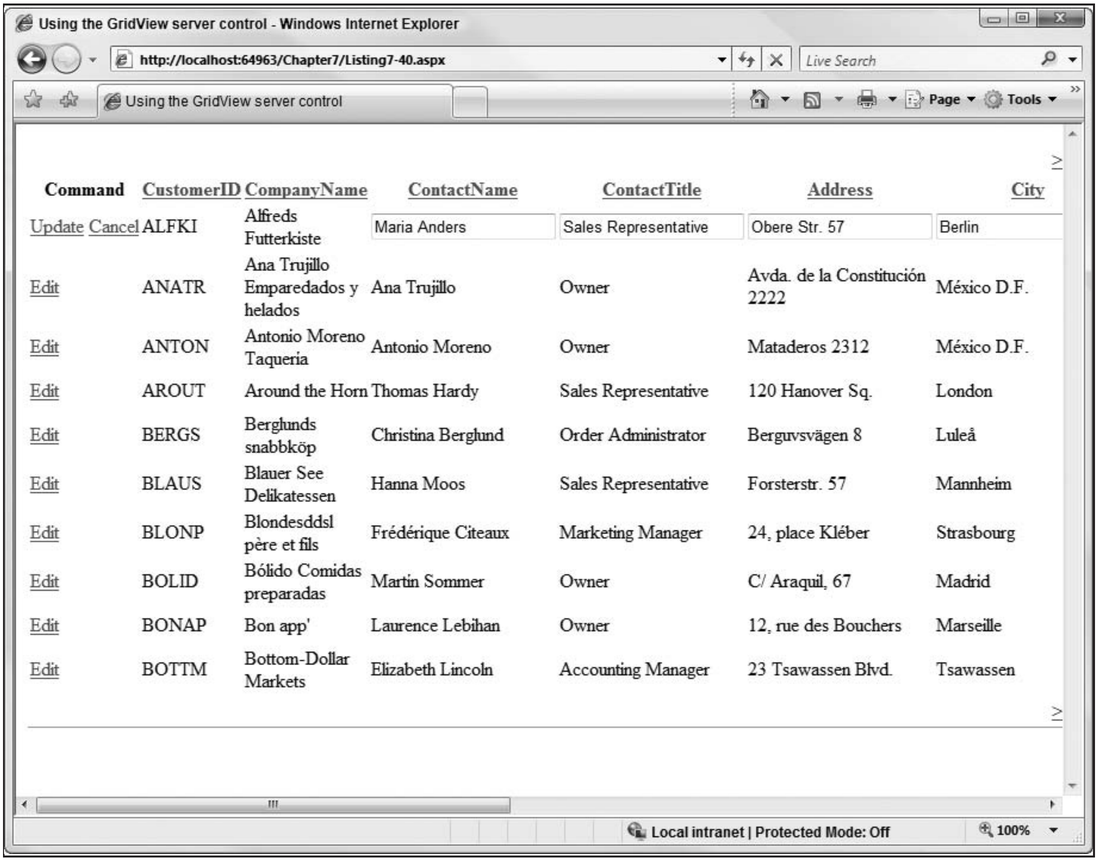
```
<asp:BoundField ReadOnly="True" HeaderText="CustomerID" DataField="CustomerID"
    SortExpression="CustomerID" Visible="False"></asp:BoundField>
```

Now if you browse to the Web page again and click the Edit button, you should see that the ID column is not editable. This is shown in Figure 7-26.

Handling Errors When Updating Data

As much as you try to prevent them, errors happen when you save data. If you allow your users to update data in your GridView control, you should implement a bit of error handling to make sure errors do not bubble up to the user.

To check for errors when updating data through the GridView, you can use the `RowUpdated` event. Listing 7-41 shows how to check for errors after a user has attempted to update data. In this scenario, if an error does occur, you simply display a message to the user in a Label.


Using the GridView server control - Windows Internet Explorer
http://localhost:64963/Chapter7/Listing7-40.aspx
Live Search
Using the GridView server control
Page
Tools
>
Local intranet | Protected Mode: Off
100%

Command	CustomerID	CompanyName	ContactName	ContactTitle	Address	City
Update Cancel	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin
Edit	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.
Edit	ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	México D.F.
Edit	AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London
Edit	BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå
Edit	BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim
Edit	BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg
Edit	BOLID	Bólido Comidas preparadas	Martin Sommer	Owner	C/ Araquil, 67	Madrid
Edit	BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille
Edit	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen

Figure 7-26

Listing 7-41: Checking for Update errors using the RowUpdated event

VB

```
<script runat="server">
    Protected Sub GridView1_RowUpdated(ByVal sender As Object, _
       ByVal e As System.Web.UI.WebControls.GridViewUpdatedEventArgs)

        If e.Exception IsNot Nothing Then
            Me.lblErrorMessage.Text = e.Exception.Message
        End If
    End Sub
</script>
```

C#

```
<script runat="server">
    protected void GridView1_RowUpdated(object sender, GridViewUpdatedEventArgs e)
    {
        if (e.Exception != null)
        {
            this.lblErrorMessage.Text = e.Exception.Message;
        }
    }
</script>
```

Using the TemplateField's EditItemTemplate

Earlier in the chapter, you were introduced to the TemplateField and various other available templates. One of those templates is the EditItemTemplate, which the grid uses to display the TemplateField column for a row that has entered edit mode. Although the standard BoundField allows users to edit their data only in text boxes, the TemplateField's EditItemTemplate enables you to completely customize the data editing experience of the user. For instance, the user would probably have a better editing experience if the Region column were presented as a DropDownList control during edit mode, rather than as a simple text box.

To do this, you simply change the Region column from a BoundField to a TemplateField and add an ItemTemplate and an EditItemTemplate. In the EditItemTemplate, you can add a DropDownList control and provide the proper data-binding information so that the control is bound to a unique list of Regions. Listing 7-42 shows how you can add the ItemTemplate and EditItemTemplate to the GridView.

Listing 7-42: Adding the ItemTemplate and EditItemTemplate to the GridView

```
<asp:TemplateField HeaderText="Country">
    <ItemTemplate><%# Eval("Country") %></ItemTemplate>
    <EditItemTemplate>
        <asp:DropDownList ID="DropDownList1" runat="server"
            DataSourceID="SqlDataSource2"
            DataTextField="Country" DataValueField="Country">
        </asp:DropDownList>
        <asp:SqlDataSource ID="SqlDataSource2" runat="server"
            ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
            SelectCommand="SELECT DISTINCT [Country] FROM [Customers]">
        </asp:SqlDataSource>
    </EditItemTemplate>
</asp:TemplateField>
```

Notice that you use a simple Eval data-binding expression in the ItemTemplate to display the value of the column in the row's default display mode. In the EditItemTemplate, you simply include a DropDownList control and a SqlDataSource control for the drop-down list to bind to.

To select the current Country value in the DropDownList control, you use the `RowDataBound` event. Listing 7-43 shows how this is done.

Listing 7-43: Using RowDataBound

VB

```
Protected Sub GridView1_RowDataBound(ByVal sender As Object, _
    ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs)

    'Check for a row in edit mode.
    If ((e.Row.RowState = DataControlRowState.Edit) Or _
```

Continued

```
            (e.Row.RowState = (DataControlRowState.Alternate Or _
            DataControlRowState.Edit))) Then

            Dim drv As System.Data.DataRowView = _
                CType(e.Row.DataItem, System.Data.DataRowView)

            Dim ddl As DropDownList = _
                CType(e.Row.Cells(8).FindControl("DropDownList1"), DropDownList)
            Dim li As ListItem = ddl.Items.FindByValue(drv("Country").ToString())
            li.Selected = True
        End If
    End Sub
```

C#

```
protected void GridView1_RowDataBound(object sender, GridViewRowEventArgs e)
{
    // Check for a row in edit mode.
    if ( (e.Row.RowState == DataControlRowState.Edit) ||
         (e.Row.RowState == (DataControlRowState.Alternate |
                             DataControlRowState.Edit)) )
    {
        System.Data.DataRowView drv = (System.Data.DataRowView)e.Row.DataItem;

        DropDownList ddl =
            (DropDownList)e.Row.Cells[8].FindControl("DropDownList1");
        ListItem li = ddl.Items.FindByValue(drv["Country"].ToString());
        li.Selected = true;
    }
}
```

As shown in the listing, to set the appropriate DropDownList value, first check that the currently bound GridViewRow is in an edit state by using the `RowState` property. The `RowState` property is a bitwise combination of DataControlRowState values. The following table shows you the possible states for a GridViewRow.

RowState	Description
Alternate	Indicates that this row is an alternate row.
Edit	Indicates the row is currently in edit mode.
Insert	Indicates the row is a new row, and is currently in insert mode.
Normal	Indicates the row is currently in a normal state.
Selected	Indicates the row is currently the selected row in the GridView.

Note that in order to determine the current RowState correctly in the previous listing, you must make multiple comparisons against the `RowState` property. The RowState can be in multiple states at once — for example, alternate and edit. Therefore, you need to use a bitwise comparison to properly determine if the GridViewRow is in an edit state. After the row is determined to be in an edit state, locate the DropDownList control in the proper GridViewRow cell by using the `FindControl` method. This method allows you to locate a server control by name. After you find the DropDownList control, locate the appropriate DropDownlist ListItem and set its `Selected` property to `True`.

You also need to use a GridView event to add the value of the DropDownList control back into the GridView after the user updates the row. For this, you can use the `RowUpdating` event as shown in Listing 7-44.

Listing 7-44: Using RowUpdating

VB

```
Protected Sub GridView1_RowUpdating(ByVal sender As Object, _
    ByVal e As System.Web.UI.WebControls.GridViewUpdateEventArgs)

    Dim gvr As GridViewRow = Me.GridView1.Rows(Me.GridView1.EditIndex)
    Dim ddl As DropDownList = _
        CType(gvr.Cells(8).FindControl("DropDownList1"), DropDownList)
    e.NewValues("Country") = ddl.SelectedValue
End Sub
```

C#

```
protected void GridView1_RowUpdating(object sender, GridViewUpdateEventArgs e)
{
    GridViewRow gvr = this.GridView1.Rows[this.GridView1.EditIndex];
    DropDownList ddl = (DropDownList)gvr.Cells[8].FindControl("DropDownList1");
    e.NewValues["Country"] = ddl.SelectedValue;
}
```

In this event, you determine GridViewRow that is currently being edited using the `EditIndex`. This property contains the index of the GridViewRow that is currently in an edit state. After you find the row, locate the DropDownList control in the proper row cell using the `FindControl` method, as in the previous listing. After you find the DropDownList control, simply add the `SelectedValue` of that control to the GridView controls `NewValues` collection.

Deleting GridView Data

Deleting data from the table produced by the GridView is even easier than editing data. Just a few additions to the code enable you to delete an entire row of data from the table. Much like with the Edit buttons you added earlier, you can easily add a Delete button to the grid by setting the `AutoGenerateDeleteButton` property to `True`. This is shown in Listing 7-45.

Listing 7-45: Adding a delete link to the GridView

```
<asp:GridView ID="GridView1" Runat="server" DataSourceID="SqlDataSource1"
    DataKeyNames="CustomerID" AutoGenerateColumns="False"
    AllowSorting="True" AllowPaging="True"
    AutoGenerateEditButton="true" AutoGenerateDeleteButton="true">
```

The addition of the `AutoGenerateDeleteButton` attribute to the GridView is the only change you make to this control. Now look at the SqlDataSource control. Listing 7-46 shows you the root element of this control.

Listing 7-46: Adding delete functionality to the SqlDataSource Control

```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server"
    SelectCommand="SELECT * FROM [Customers]"
    ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>"
    DataSourceMode="DataSet"
    DeleteCommand="DELETE From Customers WHERE (CustomerID = @CustomerID)"
    UpdateCommand="UPDATE [Customers] SET [CompanyName] = @CompanyName,
        [ContactName] = @ContactName, [ContactTitle] = @ContactTitle,
        [Address] = @Address, [City] = @City, [Region] = @Region,
        [PostalCode] = @PostalCode, [Country] = @Country, [Phone] = @Phone,
        [Fax] = @Fax WHERE [CustomerID] = @original_CustomerID">
```

In addition to the `SelectCommand` and `UpdateCommand` attributes, you also add the `DeleteCommand` attribute to the SqlDataSource and provide the SQL command that deletes the specified row. Just like the `UpdateCommand` property, the `DeleteCommand` property makes use of named parameters. Because of this, you define this parameter from within the SqlDataSource control. To do this, add a `<DeleteParameters>` section to the SqlDataSource control. This is shown in Listing 7-47.

Listing 7-47: Adding a <DeleteParameters> section to the SqlDataSource control

```
<DeleteParameters>
   <asp:Parameter Name="CustomerID" Type="String">
   </asp:Parameter>
</DeleteParameters>
```

This is the only parameter definition needed for the `<DeleteParameters>` section because the SQL command for this deletion requires only the CustomerID from the row to delete the entire row.

When you run the example with this code in place, you see a Delete link next to the Edit link. Clicking the Delete link completely deletes the selected row. Remember that it is a good idea to check for database errors after you complete the deletion. Listing 7-48 shows how you can use the GridView's `RowDeleted` event and the SqlDataSources `Deleted` event to check for errors that might have occurred during the Delete.

Notice that both events provide Exception properties to you as part of the event arguments. If the properties are not empty, then an exception occurred that you can handle. If you do choose to handle the exception, then you should set the `ExceptionHandled` property to `True`; otherwise, the exception will continue to bubble up to the end user.

Listing 7-48: Using the RowDeleted event to catch SQL errors

VB

```
<script runat="server">
    Protected Sub GridView1_RowDeleted(ByVal sender As Object, _
       ByVal e As GridViewDeletedEventArgs)

        If (Not IsDBNull (e.Exception)) Then
            Me.lblErrorMessage.Text = e.Exception.Message
            e.ExceptionHandled = True
        End If
    End Sub

    Protected Sub SqlDataSource1_Deleted(ByVal sender As Object, _
        ByVal e As System.Web.UI.WebControls.SqlDataSourceStatusEventArgs)

        If (e.Exception IsNot Nothing) Then
            Me.lblErrorMessage.Text = e.Exception.Message
            e.ExceptionHandled = True
        End If
    End Sub
</script>
```

C#

```
<script runat="server">
    protected void GridView1_RowDeleted(object sender, GridViewDeletedEventArgs e)
    {
        if (e.Exception != null)
        {
            this.lblErrorMessage.Text = e.Exception.Message;
            e.ExceptionHandled = true;
        }
    }

    protected void SqlDataSource1_Deleted(object sender,
        SqlDataSourceStatusEventArgs e)
    {
        if (e.Exception != null)
        {
            this.lblErrorMessage.Text = e.Exception.Message;
            e.ExceptionHandled = true;
        }
    }
</script>
```

Other GridView Formatting Features

The GridView control includes numerous other properties that let you adjust the look and feel of the control in fine detail. The `Caption` property allows you to set a caption at the top of the grid. The `ShowHeader` and `ShowFooter` properties enable you to control whether the column headers or footers are shown. The control also includes eight different *style properties* that give you control over the look and feel of different parts of the grid. The following table describes the style properties.

Style Property	Description
`AlternatingRowStyle`	Style applied to alternating GridView rows
`EditRowStyle`	Style applied to a GridView row in edit mode
`EmptyDataRowStyle`	Style applied to the EmptyDataRow when there are datarows available for the grid to bind to
`FooterStyle`	Style applied to the footer of the GridView
`HeaderStyle`	Style applied to the header of the GridView

Style Property	Description
`PagerStyle`	Style applied to the GridView pager
`RowStyle`	Style applied to the default GridView row
`SelectedRowStyle`	Style applied to the currently selected GridView row

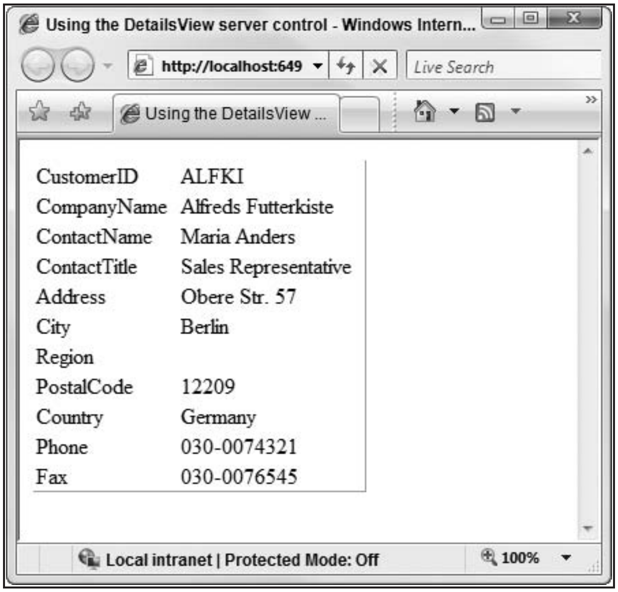
These style properties let you set the font, forecolor, backcolor, alignment, and many other style-related properties for these individual areas of the grid.

The GridView smart tag also includes an AutoFormat option that enables you to select from a list of predefined styles to apply to the control.

DetailsView

The DetailsView server control is a new data-bound control that enables you to view a single data record at a time. Although the GridView control is an excellent control for viewing a collection of data, many scenarios demand that you be able to drill down into an individual record. The DetailsView control allows you to do this and provides many of the same data manipulation and display capabilities as the GridView. It allows you to do things such as paging, updating, inserting, and deleting data.

To start using the DetailsView, drag the control onto the design surface. Like the GridView, you can use the DetailsView's smart tag to create and set the data source for the control. For this sample, just use the SqlDataSource control you used for the GridView. If you run the page at this point, you see that the control displays one record, the first record returned by your query. Figure 7-27 shows you what the DetailsView looks like in a Web page.


Using the DetailsView server control - Windows Intern...
http://localhost:649
Live Search
Using the DetailsView ...
Local intranet | Protected Mode: Off
100%

CustomerID	ALFKI
CompanyName	Alfreds Futterkiste
ContactName	Maria Anders
ContactTitle	Sales Representative
Address	Obere Str. 57
City	Berlin
Region	
PostalCode	12209
Country	Germany
Phone	030-0074321
Fax	030-0076545

Figure 7-27

If this is all the functionality you want, you probably want to create a new SqlDataSource control and modify the `SelectCommand` so that it returns only one record, rather than returning all records as our query does. For this example, however, you want to be able to page through all the customer data returned by your query. To do this, simply turn on paging by setting the DetailsView's `AllowPaging` property to `True`. You can either check the Enable Paging check box in the DetailsView smart tag or add the attribute to the control in HTML View. Listing 7-49 shows the DetailsView code for the control.

Listing 7-49: Enabling paging on the DetailsView control

```
<asp:DetailsView ID="DetailsView1" Runat="server" DataSourceID="SqlDataSource1"
     AutoGenerateRows="False" DataKeyNames="CustomerID"></asp:DetailsView>
```

Like the GridView, the DetailsView control enables you to customize the control's Pager using the `PagerSettings-Mode`, as well as the `Pager` style.

Customizing the DetailsView Display

You can customize the appearance of the DetailsView control by picking and choosing which fields the control displays. By default, the control displays each column from the table it is working with. Much like the GridView control, however, the DetailsView control enables you to specify that only certain selected columns be displayed, as illustrated in Listing 7-50.

Listing 7-50: Customizing the display of the DetailsView control

```
<asp:DetailsView ID="DetailsView1" Runat="server" DataSourceID="SqlDataSource1"
     AutoGenerateRows="False" DataKeyNames="CustomerID">
      <Fields>
           <asp:BoundField ReadOnly="True" HeaderText="CustomerID"
                DataField="CustomerID" SortExpression="CustomerID"
                Visible="False" />
           <asp:BoundField ReadOnly="True" HeaderText="CompanyName"
                DataField="CompanyName" SortExpression="CompanyName" />
           <asp:BoundField HeaderText="ContactName" DataField="ContactName"
                SortExpression="ContactName" />
           <asp:BoundField HeaderText="ContactTitle" DataField="ContactTitle"
                SortExpression="ContactTitle" />
       </Fields>
</asp:DetailsView>
```

Using the DetailsView and GridView Together

This section looks at a common scenario using both the GridView and the DetailsView. In this example, you use the GridView to display a master view of the data and the DetailsView to show the details of the selected GridView row. The Customers table is the data source. Listing 7-51 shows the code needed for this.

Listing 7-51: Using the GridView and DetailsView together

```
 <html>
<head id="Head1" runat="server">
    <title>GridView & DetailsView Controls</title>
</head>
```

Continued

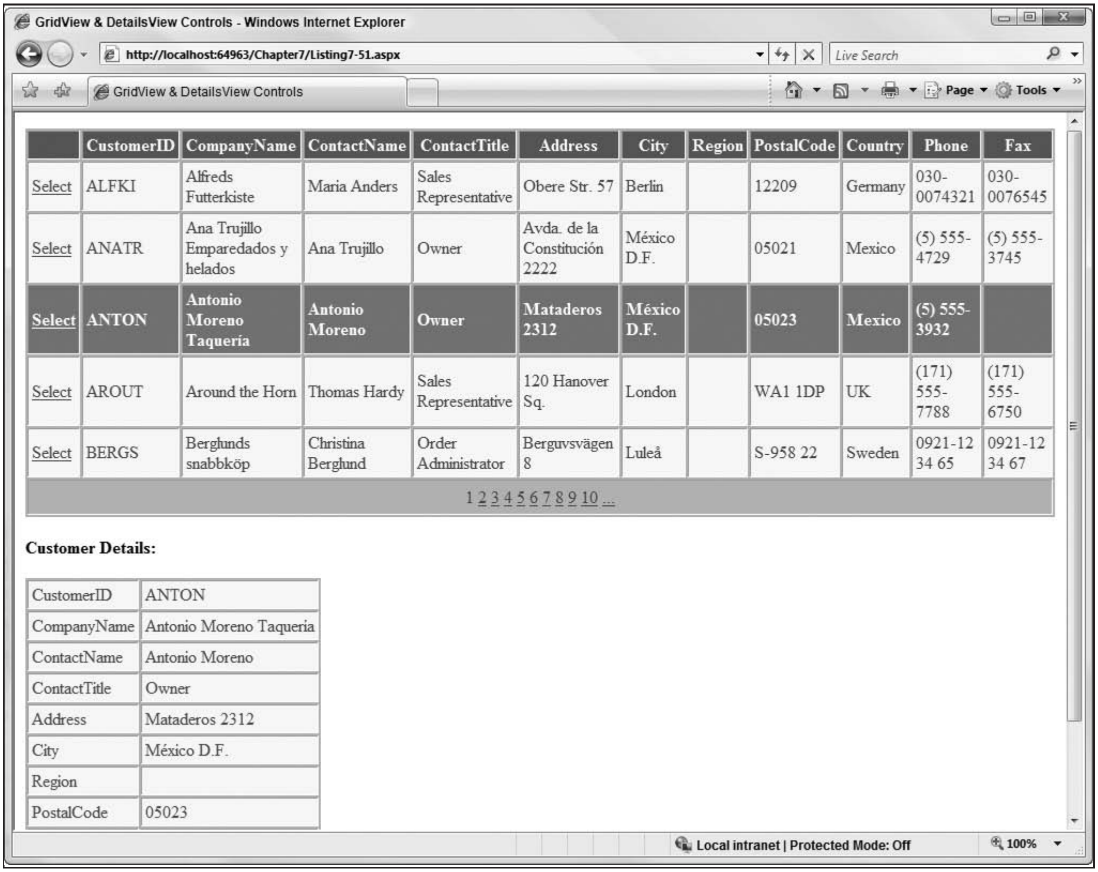
```
<body>
    <form id="form1" runat="server">
        <p>
            <asp:GridView ID="GridView1" runat="server"
                 DataSourceId="SqlDataSource1" AllowPaging="True"
                 BorderColor="#DEBA84" BorderStyle="None" BorderWidth="1px"
                 BackColor="#DEBA84" CellSpacing="2" CellPadding="3"
                 DataKeyNames="CustomerID" AutoGenerateSelectButton="True"
                 AutoGenerateColumns="False" PageSize="5">
                 <FooterStyle ForeColor="#8C4510"
                     BackColor="#F7DFB5"></FooterStyle>
                 <PagerStyle ForeColor="#8C4510"
                     HorizontalAlign="Center"></PagerStyle>
                 <HeaderStyle ForeColor="White" BackColor="#A55129"
                     Font-Bold="True"></HeaderStyle>
                 <Columns>
                     <asp:BoundField ReadOnly="True" HeaderText="CustomerID"
                          DataField="CustomerID" SortExpression="CustomerID">
                     </asp:BoundField>
                     <asp:BoundField HeaderText="CompanyName"
                          DataField="CompanyName" SortExpression="CompanyName">
                     </asp:BoundField>
                     <asp:BoundField HeaderText="ContactName"
                          DataField="ContactName" SortExpression="ContactName">
                     </asp:BoundField>
                     <asp:BoundField HeaderText="ContactTitle"
                          DataField="ContactTitle" SortExpression="ContactTitle">
                     </asp:BoundField>
                     <asp:BoundField HeaderText="Address" DataField="Address"
                          SortExpression="Address"></asp:BoundField>
                     <asp:BoundField HeaderText="City" DataField="City"
                          SortExpression="City"></asp:BoundField>
                     <asp:BoundField HeaderText="Region" DataField="Region"
                          SortExpression="Region"></asp:BoundField>
                     <asp:BoundField HeaderText="PostalCode" DataField="PostalCode"
                          SortExpression="PostalCode"></asp:BoundField>
                     <asp:BoundField HeaderText="Country" DataField="Country"
                          SortExpression="Country"></asp:BoundField>
                     <asp:BoundField HeaderText="Phone" DataField="Phone"
                          SortExpression="Phone"></asp:BoundField>
                     <asp:BoundField HeaderText="Fax" DataField="Fax"
                          SortExpression="Fax"></asp:BoundField>
                </Columns>
                <SelectedRowStyle ForeColor="White" BackColor="#738A9C"
                     Font-Bold="True"></SelectedRowStyle>
                <RowStyle ForeColor="#8C4510" BackColor="#FFF7E7"></RowStyle>
            </asp:GridView>
        </p>
        <p><b>Customer Details:</b></p>
        <asp:DetailsView ID="DetailsView1" runat="server"
             DataSourceId="SqlDataSource2"
             BorderColor="#DEBA84" BorderStyle="None" BorderWidth="1px"
             BackColor="#DEBA84" CellSpacing="2" CellPadding="3"
             AutoGenerateRows="False" DataKeyNames="CustomerID">
```

Continued

```
            <FooterStyle ForeColor="#8C4510" BackColor="#F7DFB5"></FooterStyle>
            <RowStyle ForeColor="#8C4510" BackColor="#FFF7E7"></RowStyle>
           <PagerStyle ForeColor="#8C4510" HorizontalAlign="Center"></PagerStyle>
            <Fields>
               <asp:BoundField ReadOnly="True" HeaderText="CustomerID"
                     DataField="CustomerID" SortExpression="CustomerID">
               </asp:BoundField>
               <asp:BoundField HeaderText="CompanyName" DataField="CompanyName"
                    SortExpression="CompanyName"></asp:BoundField>
               <asp:BoundField HeaderText="ContactName" DataField="ContactName"
                    SortExpression="ContactName"></asp:BoundField>
               <asp:BoundField HeaderText="ContactTitle" DataField="ContactTitle"
                    SortExpression="ContactTitle"></asp:BoundField>
               <asp:BoundField HeaderText="Address" DataField="Address"
                    SortExpression="Address"></asp:BoundField>
               <asp:BoundField HeaderText="City" DataField="City"
                    SortExpression="City"></asp:BoundField>
               <asp:BoundField HeaderText="Region" DataField="Region"
                    SortExpression="Region"></asp:BoundField>
               <asp:BoundField HeaderText="PostalCode" DataField="PostalCode"
                    SortExpression="PostalCode"></asp:BoundField>
               <asp:BoundField HeaderText="Country" DataField="Country"
                    SortExpression="Country"></asp:BoundField>
               <asp:BoundField HeaderText="Phone" DataField="Phone"
                    SortExpression="Phone"></asp:BoundField>
               <asp:BoundField HeaderText="Fax" DataField="Fax"
                    SortExpression="Fax"></asp:BoundField>
           </Fields>
           <HeaderStyle ForeColor="White" BackColor="#A55129"
               Font-Bold="True"></HeaderStyle>
           <EditRowStyle ForeColor="White" BackColor="#738A9C"
               Font-Bold="True"></EditRowStyle>
       </asp:DetailsView>
       <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            SelectCommand="SELECT * FROM [Customers]"
            ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>" />
       <asp:SqlDataSource ID="SqlDataSource2" runat="server"
            SelectCommand="SELECT * FROM [Customers]"
            FilterExpression="CustomerID='{0}'"
            ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>">
            <FilterParameters>
                <asp:ControlParameter Name="CustomerID" ControlId="GridView1"
                     PropertyName="SelectedValue"></asp:ControlParameter>
            </FilterParameters>
       </asp:SqlDataSource>
   </form>
</body>
</html>
```

When this code is run in your browser, you get the results shown in Figure 7-28.

In this figure, one of the rows in the GridView has been selected (noticeable because of the gray highlighting). The details of the selected row are shown in the DetailsView control directly below the GridView control.


GridView & DetailsView Controls - Windows Internet Explorer
http://localhost:64963/Chapter7/Listing7-51.aspx
Live Search
GridView & DetailsView Controls
Page
Tools
Local intranet | Protected Mode: Off
100%

	CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
Select	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076545
Select	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555-3745
Select	ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	
Select	AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK	(171) 555-7788	(171) 555-6750
Select	BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65	0921-12 34 67

1 2 3 4 5 6 7 8 9 10 ...

Customer Details:

CustomerID	ANTON
CompanyName	Antonio Moreno Taqueria
ContactName	Antonio Moreno
ContactTitle	Owner
Address	Mataderos 2312
City	México D.F.
Region	
PostalCode	05023

Figure 7-28

To see how this works, look at the changes that were made to the second SqlDataSource control, SqlDataSource2. Notice that a `FilterExpression` attribute has been added, which is used to filter the data reterived by the `SelectCommand`.

The value given to the `FilterExpression` attribute expresses how you want the SqlDataSource control to filter its `Select` command. In this case, the value of the `FilterExpression` is `CustomerID='0'`. This tells the SqlDataSource control to filter records that it returns by the CustomerID given to it, as shown in Listing 7-52. The `FilterExpression` attribute uses the same syntax as `String.Format`.

Listing 7-52: Filtering SqlDataSource data with a FilterExpression

```
<asp:SqlDataSource ID="SqlDataSource2" runat="server"
    SelectCommand="SELECT * FROM [Customers]"
    FilterExpression="CustomerID='{0}'"
    ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>">
    <FilterParameters>
        <asp:ControlParameter Name="CustomerID" ControlId="GridView1"
```

```
                PropertyName="SelectedValue"></asp:ControlParameter>
        </FilterParameters>
    </asp:SqlDataSource>
```

The parameter specified in the `FilterExpression` attribute, `CustomerID`, is defined within the SqlDataSource control through the use of the `<FilterParameters>` element. This sample uses an `<asp:ControlParameter>` to specify the name of the parameter, the control that the parameter value is coming from (the GridView control), and the property name that is used to populate the parameter value.

Finally, be sure to include the `DataKeyNames` attribute in the GridView control. In this case supply `CustomerID` as the value. This tells the GridView which column(s) are to be used as a primary key. When a user selects a row, the value of that column is then provided to the DetailsView control via the `SelectValue` property. The procedure for adding the `DataKeyNames` attribute to the GridView is shown in Listing 7-53.

Listing 7-53: Adding the DataKeyNames attribute to the GridView

```
<asp:GridView ID="GridView1" runat="server"
             DataSourceId="SqlDataSource1" AllowPaging="True"
             BorderColor="#DEBA84" BorderStyle="None" BorderWidth="1px"
             BackColor="#DEBA84" CellSpacing="2" CellPadding="3"
             DataKeyNames="CustomerID" AutoGenerateSelectButton="True"
             AutoGenerateColumns="False" PageSize="5">
```

SelectParameters versus FilterParameters

You might have noticed in the previous example that the `FilterParameters` seem to provide the same functionality as the `SelectParameters`. Although both produce essentially the same result, they use very different methods. As you saw in the section ''Filtering Data Using SelectParameters,'' using the `SelectParameters` allows the developer to inject values into a `WHERE` clause specified in the `SelectCommand`. This limits the rows that are returned from the SQL Server and held in memory by the data source control. The advantage is that by limiting the amount of data returned from SQL, you can make your application faster and reduce the amount of memory it consumes. The disadvantage is that you are confined to working with the limited subset of data returned by the SQL query.

`FilterParameters`, on the other hand, do not need to use a `WHERE` clause in the `SelectCommand`, requiring all the data to be returned from the SQL server to the Web server. The filter is applied to the data source control's in-memory data, rather than using a `WHERE` clause, which would have allowed SQL Server to limit the results it returns. The disadvantage of the filter method is that more data has to be transferred to the Web server. However, in some cases this is an advantage such as when you are performing many filters of one large chunk of data (for instance, to enable paging in the DetailView), you do not have to call out to SQL Server each time you need the next record. All the data is stored in cache memory by the data source control.

Inserting, Updating, and Deleting Data Using DetailsView

Inserting data using the DetailsView is similar to all the other data functions that you have performed. To insert data using the DetailsView, simply add the `AutoGenerateInsertButton` attribute to the DetailsView control as shown in Listing 7-54.

Listing 7-54: Adding an AutoGenerateInsertButton attribute to the DetailsView

```
<asp:DetailsView ID="DetailsView1" runat="server"
     DataSourceId="SqlDataSource2"
     AutoGenerateRows="False" AutoGenerateInsertButton="true"
     DataKeyNames="CustomerID">
```

Then add the `InsertCommand` and corresponding `InsertParameter` elements to the SqlDataSource control, as shown in Listing 7-55.

Listing 7-55: Adding an InsertCommand to the SqlDataSource control

```
<asp:SqlDataSource ID="sqlDataSource2" runat="server"
     SelectCommand="SELECT * FROM [Customers]"
     InsertCommand="INSERT INTO [Customers] ([CustomerID], [CompanyName],
         [ContactName], [ContactTitle], [Address], [City], [Region], [PostalCode],
         [Country], [Phone], [Fax]) VALUES (@CustomerID, @CompanyName,
         @ContactName, @ContactTitle, @Address, @City, @Region, @PostalCode,
         @Country, @Phone, @Fax)" DeleteCommand="DELETE FROM [Customers] WHERE
         [CustomerID] = @original_CustomerID"
     ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>">
     <InsertParameters>
         <asp:Parameter Type="String" Name="CustomerID"></asp:Parameter>
         <asp:Parameter Type="String" Name="CompanyName"></asp:Parameter>
         <asp:Parameter Type="String" Name="ContactName"></asp:Parameter>
         <asp:Parameter Type="String" Name="ContactTitle"></asp:Parameter>
         <asp:Parameter Type="String" Name="Address"></asp:Parameter>
         <asp:Parameter Type="String" Name="City"></asp:Parameter>
         <asp:Parameter Type="String" Name="Region"></asp:Parameter>
         <asp:Parameter Type="String" Name="PostalCode"></asp:Parameter>
         <asp:Parameter Type="String" Name="Country"></asp:Parameter>
         <asp:Parameter Type="String" Name="Phone"></asp:Parameter>
         <asp:Parameter Type="String" Name="Fax"></asp:Parameter>
     </InsertParameters>
</asp:SqlDataSource>
```

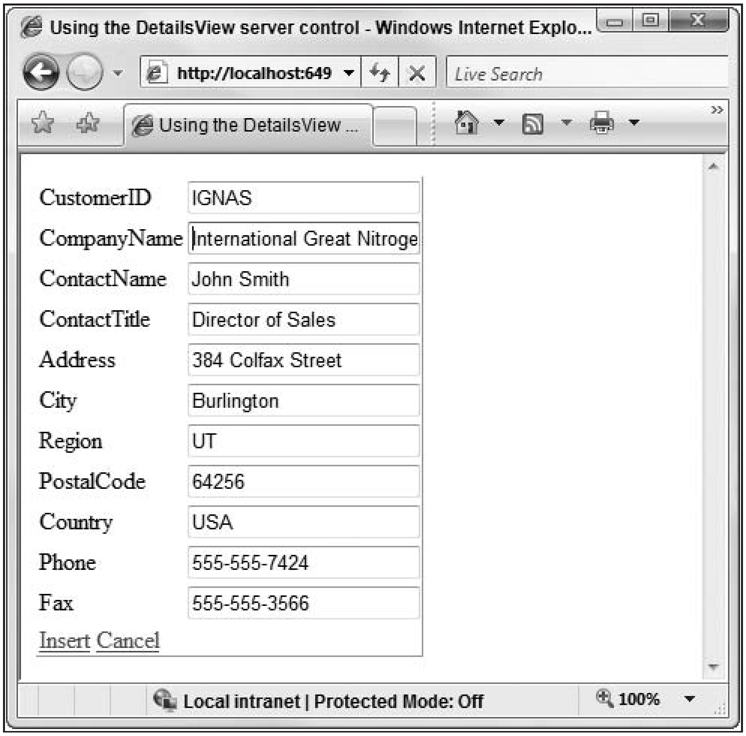
Figure 7-29 shows the DetailsView control page loaded in the browser in insert mode, ready to add a new record.

Figure 7-30 shows the DetailsView control after a new record has been inserted.

Updating and deleting data using the DetailsView control are similar to updating and deleting data from the GridView. Simply specify the `UpdateCommand` or `DeleteCommand` attributes in the DetailView control; then provide the proper `UpdateParameters` and `DeleteParameters` elements.

ListView

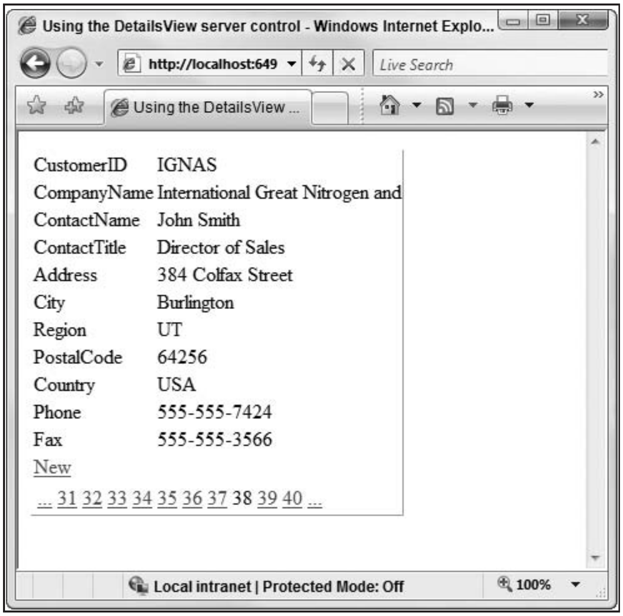
ASP.NET 3.5 introduces a new data-bound list control called the ListView. The idea behind the ListView control is to offer a data-bound control that bridges the gap between the highly structured GridView control introduced in ASP.NET 2.0, and the anything goes, unstructured controls DataList and Repeater.


Using the DetailsView server control - Windows Internet Explo...
http://localhost:649
Live Search
Using the DetailsView ...
Local intranet | Protected Mode: Off
100%

CustomerID	IGNAS
CompanyName	International Great Nitroge
ContactName	John Smith
ContactTitle	Director of Sales
Address	384 Colfax Street
City	Burlington
Region	UT
PostalCode	64256
Country	USA
Phone	555-555-7424
Fax	555-555-3566

Insert Cancel

Figure 7-29


Using the DetailsView server control - Windows Internet Explo...
http://localhost:649
Live Search
Using the DetailsView ...
Local intranet | Protected Mode: Off
100%

CustomerID	IGNAS
CompanyName	International Great Nitrogen and
ContactName	John Smith
ContactTitle	Director of Sales
Address	384 Colfax Street
City	Burlington
Region	UT
PostalCode	64256
Country	USA
Phone	555-555-7424
Fax	555-555-3566

New

... 31 32 33 34 35 36 37 38 39 40 ...

Figure 7-30

In the past, many developers who wanted a grid style data control chose the GridView because it was easy to use and offered powerful features such as data editing, paging, and sorting. Unfortunately, the more developers dug into the control, the more they found that controlling the way it rendered its HTML output was exceedingly difficult. This was problematic if you wanted to lighten the amount of markup generated by the control, or use CSS exclusively to control the control's layout and style.

On the other side of the coin, many developers were drawn the DataList or Repeater because of the enhanced control they got over rendering. These controls contained little to no notion of layout and required the developer to start from scratch in laying out their data. Unfortunately, these controls lacked some of the basic features of the GridView, such as paging and sorting, or in the case of the Repeater, any notion of data editing.

This is where the ListView attempts to fill the gap between these controls. The control itself emits no runtime generated HTML markup; instead it relies on a series of 11 different control templates that represent the different areas of the control and the possible states of those areas. Within these templates you can place markup auto-generated by the control at design-time, or markup created by the developer, but in either case the developer retains complete control over not only the markup for individual data items in the control, but of the markup for the layout of the entire control. Additionally, because the control readily understands and handles data editing and paging, you can let the control do much of the data management work, allowing you to focus primarily on the display of your data.

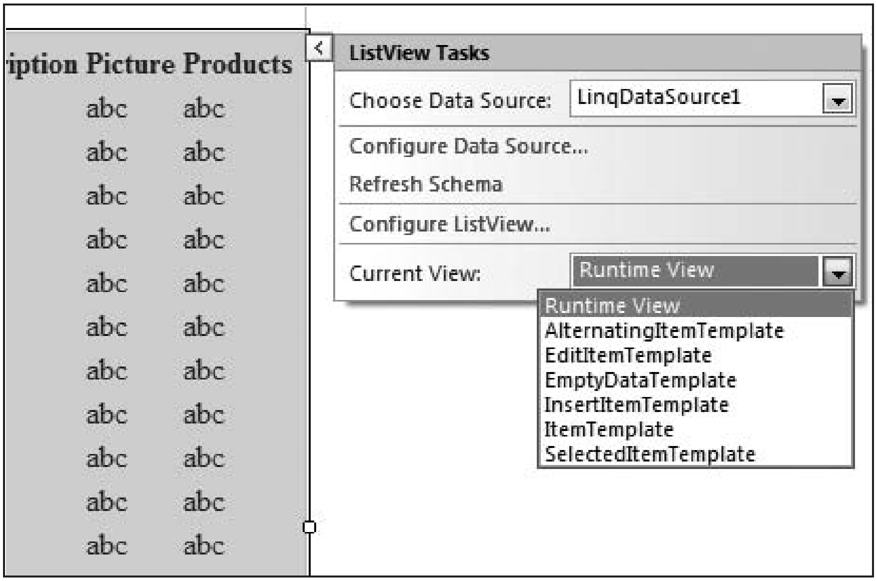
Getting Started with the ListView

To get started using the ListView, simply drop the control on the design surface and assign a data source to it just as you would any other data-bound list control. Once you assign the data source, however, you will see that there is no design-time layout preview available as you might expect. This is because, by default, the ListView has no layout defined and it is completely up to you to define the control's layout. In fact, the design-time rendering of the control even tells you that you need to define at least an ItemTemplate and LayoutTemplate in order to use the control. The LayoutTemplate serves as the root template for the control, and the ItemTemplate serves as the template for each individual data item in the control.

You have two options for defining the templates needed by the ListView. You can either edit the templates directly by changing the Current View option in the ListView smart tag, or you can select a predefined layout from the controls smart tag. Changing Current View allows you to see a runtime view of each of the available templates, and edit the contents of those templates directly just as you would normally edit any other control template. Figure 7-31 shows the Current View drop-down in the ListView's smart tag.

The second option and probably the easier to start with, is to choose a predefined layout template from the Configure ListView dialog. To open this dialog, simply click the ConfigureListView option from the smart tag. Once open, you are presented with a dialog that lets you select between several different pre-defined layouts, select different style options, and even configure basic behavior options such as editing and paging. This dialog is shown in Figure 7-32.

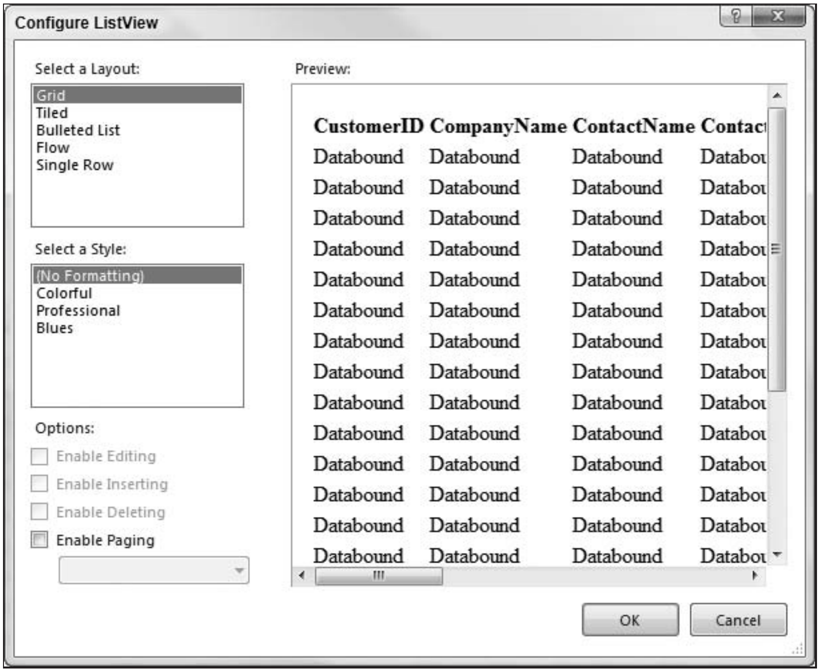
The control includes five different layout types: Grid, Tiled, Bulleted List, Flow, and Single Row and four different style options. A preview of each type is presented in the dialog, and as you change the currently selected layout and style, the preview is updated.


iption Picture Products
abc abc
ListView Tasks
Configure Data Source...
Refresh Schema
Configure ListView...
Runtime View
AlternatingItemTemplate
EditItemTemplate
EmptyDataTemplate
InsertItemTemplate
ItemTemplate
SelectedItemTemplate

Choose Data Source: LinqDataSource1

Current View: Runtime View

Figure 7-31


Configure ListView
Select a Layout:
Grid
Tiled
Bulleted List
Flow
Single Row
Select a Style:
(No Formatting)
Colorful
Professional
Blues
Options:
Enable Editing
Enable Inserting
Enable Deleting
Enable Paging
Preview:
CustomerID CompanyName ContactName Contact
Databound Databound Databound Databou
OK
Cancel

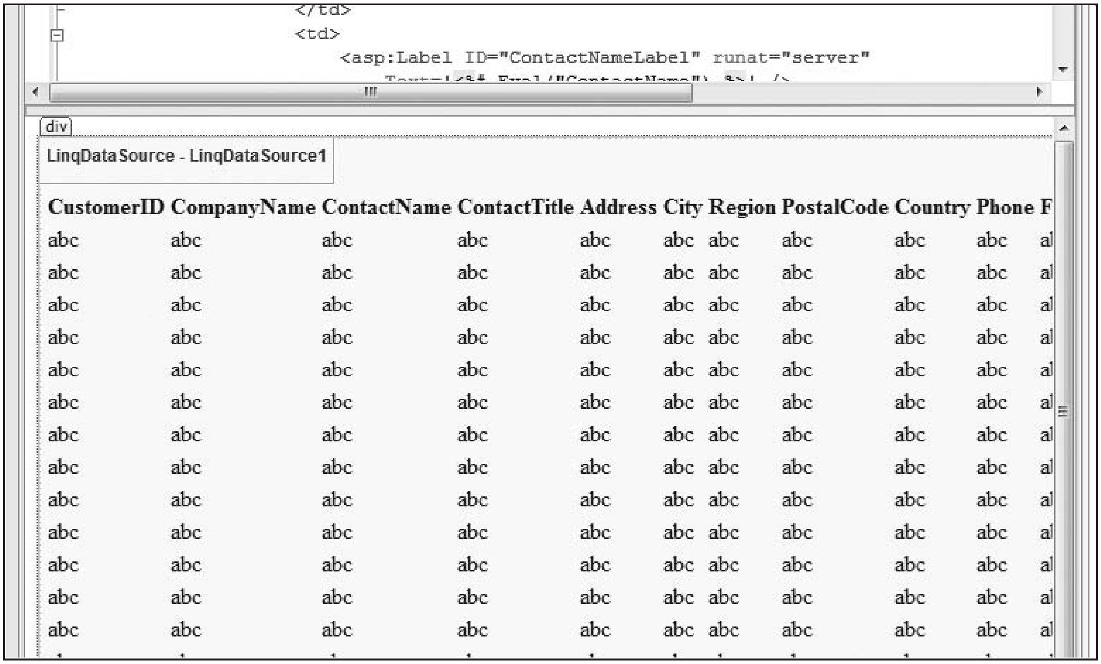
Select a Layout: Grid

Select a Style: (No Formatting)

☐ Enable Editing
☐ Enable Inserting
☐ Enable Deleting
☐ Enable Paging

Figure 7-32

To see exactly how the control defines each layout option, select the Grid layout with the Colorful style and enable Inserting, Editing, Deleting, and Paging. Click OK to apply your choices and close the dialog box. When the dialog box closes, you should now see that you get a design-time preview of your layout and running the page results in the ListView generating a grid layout, as shown in Figure 7-33.



```
                    </td>
                    <td>
                        <asp:Label ID="ContactNameLabel" runat="server"
```

div

LinqDataSource - LinqDataSource1

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	F
abc	abc	abc	abc	abc	abc	abc	abc	abc	abc	al
abc	abc	abc	abc	abc	abc	abc	abc	abc	abc	al
abc	abc	abc	abc	abc	abc	abc	abc	abc	abc	al
abc	abc	abc	abc	abc	abc	abc	abc	abc	abc	al
abc	abc	abc	abc	abc	abc	abc	abc	abc	abc	al
abc	abc	abc	abc	abc	abc	abc	abc	abc	abc	al
abc	abc	abc	abc	abc	abc	abc	abc	abc	abc	al
abc	abc	abc	abc	abc	abc	abc	abc	abc	abc	al
abc	abc	abc	abc	abc	abc	abc	abc	abc	abc	al
abc	abc	abc	abc	abc	abc	abc	abc	abc	abc	al
abc	abc	abc	abc	abc	abc	abc	abc	abc	abc	al
abc	abc	abc	abc	abc	abc	abc	abc	abc	abc	al
abc	abc	abc	abc	abc	abc	abc	abc	abc	abc	al

Figure 7-33

ListView Templates

Once you have applied a layout template to the ListView, if you look at the Source window in Visual Studio you can see that in order to provide the layout, the control actually generated a significant chunk of markup. This markup is generated based on the layout that you chose in the Configure ListView dialog.

If you closely examine the markup that has been generated for the Grid layout used in the previous section, you will see that, by default, the control creates markup for seven different control templates: the ItemTemplate, AlternatingItemTemplate, SelectedItemTemplate, InsertItemTemplate, EditItemTemplate, EmptyDataTemplate, and LayoutTemplate. These are just some of the 11 different templates that the control exposes, and that you can use to provide markup for the different states of the control. Choosing a different predefined layout option results in the control generating a different collection of templates. Of course, you can also always manually add or remove any of the templates yourself. All 11 templates are listed in the following table.

Template Name	Description
ItemTemplate	Provides a User Interface for each data item in the control
AlternatingItemTemplate	Provides a unique UI for alternating data items in the control
SelectedItemTemplate	Provides a unique UI for the currently selected data item
InsertItemTemplate	Provides a UI for inserting a new data item into the control
EditItemTemplate	Provides a UI for editing an existing data item in the control
EmptyItemTemplate	Provides a unique UI for rows created when there is no more data to display in the last group of the current page
EmptyDataTemplate	The template shown when the bound data object contains no data items
LayoutTemplate	The template that serves as the root container for the ListView control and is used to control the overall layout of the data items
GroupSeparatorTemplate	Used to provide a separator UI between groups
GroupTemplate	Used to provide a unique UI for grouped content
ItemSeperatorTemplate	Used to provide a separator UI between each data item

The use of templates allows the ListView control to retain a very basic level of information about the markup sections and states which can comprise the ListView, while still being able to give you almost total control over the UI of the ListView.

ListView Data Item Rendering

While the ListView is generally very flexible, allowing you almost complete control over the way it displays its bound data, it does have some basic structure which defines how the templates described in the previous section are related to one another. As described previously, at a minimum, the control requires you to define two templates, the LayoutTemplate and ItemTemplate. The LayoutTemplate is the root control template and therefore where you should define the overall layout for the collection of data items in the ListView.

For example, if you examine the template markup generated by the Grid layout, you can see the LayoutTemplate includes a `<table>` element definition, a single table row (`<tr>`) definition, and a `<td.>` element defined for each column header.

The ItemTemplate, on the other hand, is where you define the layout for an individual data item. If you again look at the markup generated for the Grid layout, its ItemTemplate is a single table row (`<tr>`) element followed by a series of table cell (`<td>`) elements that contain the actual data.

When the ListView renders itself, it knows that the ItemTemplate should be rendered within the LayoutTemplate, but what is needed is a mechanism to tell the control exactly where within the LayoutTemplate to place the ItemTemplate. The ListView control does this by looking within the LayoutTemplate for an

Item Container. The Item Container is an HTML container element with the `runat = "server"` attribute set and an `id` attribute whose value is `itemContainer`. The element can be any valid HTML container element, although if you examine the default Grid LayoutTemplate you will see that it uses the `<tbody>` element.

```
<tbody id="itemContainer">
</tbody>
```

Adding to the overall flexibility of the control, even the specific Item Container element `id` that ListView looks for can be configured. While by default the control will attempt to locate an element whose `id` attribute is set to `itemContainer`, you can change the `id` value the control will look for by changing the control's `ItemContainerID` property.

If the control fails to locate an appropriate HTML element designated as the Item Container, it will throw an exception.

The ListView uses the element identified as the `itemContainer` to position not only the ItemTemplate, but any item-level template, such as the AlternativeItemTemplate, EditItemTemplate, EmptyItemTemplate, InsertItemTemplate, ItemSeperatorTemplate, and SelectedItemTemplate. During rendering, it simply places the appropriate item template into the Item Container, depending on the state of the data item (selected, editing, or alternate) for each data item it is bound to.

ListView Group Rendering

In addition to the Item Container, the ListView also supports another container type, the Group Container. The Group Container works in conjunction with the GroupTemplate to allow you to divide a large group of data items into smaller sets. The number of items in each group is set by the control's `GroupItemCount` property. This is useful is when you want to output some additional HTML after some number of item templates have been rendered. When using the GroupTemplate, the same problem exists as was discussed in the prior section. In this case, however, rather than having two templates to relate, introducing the GroupTemplate means you have three templates to relate: the ItemTemplate to the GroupTemplate, and the GroupTemplate to the LayoutTemplate.

When the ListView renders itself, it looks to see if a GroupTemplate has been defined. If the control finds a GroupTemplate, then it checks to see if a Group Container is provided in the LayoutTemplate. If you have defined the GroupTemplate, then the control requires that you define a Group Container; otherwise it throws an exception. The Group Container works the same way as the Item Container described in the previous section, except that the container element's `id` value should be `groupContainer`, rather than `itemContainer`. As with Item Container, the specific `id` value the control looks for can be changed by altering the `GroupContainerID` property of the control.

You can see an example of the Group Container being used by looking at the markup generated by the ListViews Tiled layout. The LayoutTemplate of this layout shows a table serving as the Group Container, shown here:

```
<table id="groupContainer" runat="server" border="0" style="">
</table>
```

Once a GroupContainer is defined, you need to define an Item Container, but rather than doing this in the LayoutTemplate, you need to do this in the GroupTemplate. Again, looking at the Tiled layout, you can see that within its GroupTemplate, it defined a table row which serves as the Item Container.

```
<tr id="itemContainer" runat="server">
</tr>
```

When rendering, the ListView will output its LayoutTemplate first, and then output the GroupTemplate. The ItemTemplate is then output the number of times defined by the `GroupItemCount` property. Once the group item count has been reached, the ListView outputs the GroupTemplate, then ItemTemplate again, repeating this process for each data item it is bound to.

Using the EmptyItemTemplate

When using the GroupTemplate, it is also important to keep in mind that the number of data items bound to the ListView control may not be perfectly divisible by the GroupItemCount value. This is especially important to keep in mind if you have created a ListView layout that is dependent on HTML tables for its data item arrangement because there is a chance that the last row may end up defining fewer table cells than previous table rows, making the HTML output by the control invalid, and possibly causing rendering problems. To solve this, the ListView control includes the EmptyItemTemplate. This template is rendered if you are using the GroupTemplate, and there are not enough data items remaining to reach the `GroupItemCount` value. Figure 7-34 shows an example of when the EmptyItemTemplate would be used.

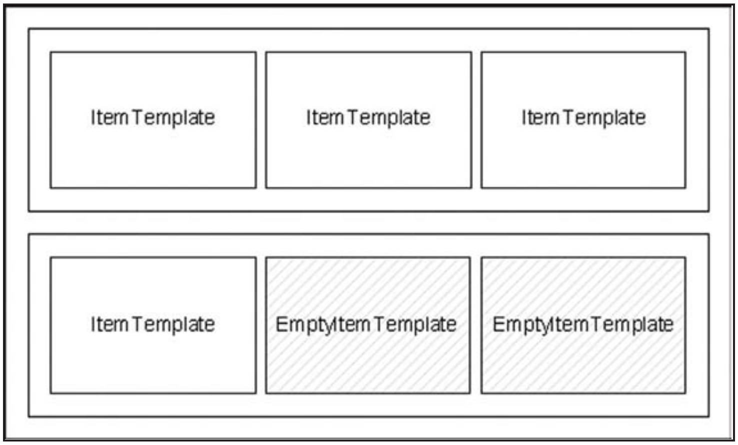


Figure 7-34

In this scenario, the data source bound to the ListView control contains four data items, but the `GroupItemCount` for the control is set to `3`, meaning there will be three ItemTemplates rendered in each group. You can see that this means for the second group rendered, there will only be a single data item remaining to render; therefore, the control will use the EmptyItemTemplate, if defined, to fill the remaining items.

You can also see another example of the use of the EmptyItemTemplate in the ListView's Tiled layout.

ListView Data Binding and Commands

Because the ListView does not generate any layout markup at runtime and does not include any of the auto field generation logic as you may be used to in the GridView, each template uses the standard ASP.NET inline data-binding syntax to position the values of each data item in the defined layout.

ASP.NET's inline data-binding syntax is covered in detail later in this chapter.

You can see this by examining the ItemTemplate of the default Grid layout created by the control. In this template, each column of the bound data source is displayed using an ASP.NET label whose text property is set to a data-binding evaluation expression:

```
<asp:Label ID="ProductNameLabel" runat="server"
    Text='<%# Eval("ProductName") %>' />
```

Because the control uses this flexible model to display the bound data, you can leverage it to place the data wherever you would like within the template, and even use the features of ASP.NET data binding to manipulate the bound data before it is displayed.

Every ListView template that displays bound data uses the same ASP.NET binding syntax, and simply provides a different template around it. For example, if you enable editing in the Grid layout you will see that the EditItemTemplate simply replaces the ASP.NET Label used by the ItemTemplate with a TextBox or Checkbox depending on the underlying data type.

```
<asp:TextBox ID="ProductNameTextBox" runat="server"
    Text='<%# Bind("ProductName") %>' />
```

Again, this flexibility allows you to choose exactly how you want to allow your end user to edit the data (if you want it to be editable). Instead of a standard ASP.NET TextBox, you could easily replace this with a DropDownList, or even a third-party editing control.

To get the ListView to show the EditItemTemplate for a data item, the control uses the same commands concept found in the GridView control. The ItemTemplate provides three commands you can use to change the state of a data item.

Command Name	Description
`Edit`	Places the specific data item into edit mode and shows the EditTemplate for the data item
`Delete`	Deletes the specific data item from the underlying data source
`Select`	Sets the ListView controls Selected index to the index of the specific data item

These commands are used in conjunction with the ASP.NET Button control's `CommandName` property. You can see these commands used in ItemTemplate of the ListViews default Grid layout by enabling Editing and Deleting using the ListView configuration dialog. Doing this generates a new column with an Edit and Delete button, each of which specified the `CommandName` property set to Edit and Delete respectively.

```
<asp:Button ID="DeleteButton" runat="server"
    CommandName="Delete" Text="Delete" />
<asp:Button ID="EditButton" runat="server"
    CommandName="Edit" Text="Edit" />
```

Other templates in the ListView offer other commands, as shown in the table that follows.

Template	Command Name	Description
EditItemTemplate	Update	Updates the data in the ListViews data source and returns the data item to the ItemTemplate display
EditItemTemplate	Cancel	Cancels the edit and returns the data item to the ItemTemplate
InsertItemTemplate	Insert	Inserts the data into the ListViews at a source
InsertItemTemplate	Cancel	Cancels the insert and resets the InsertTemplate controls binding values

ListView Paging and the Pager Control

ASP.NET 3.5 introduced another new control called the DataPager control that the ListView uses to provide paging capabilities for itself. The DataPager control is designed to display the navigation for paging to the end user and to coordinate data paging with any data bound control that implements the `IPagableItemContainer` interface, which in ASP.NET 3.5 is the ListView control. In fact, you will notice that if you enable paging on the ListView control by checking the Paging check box in the ListView configuration dialog, the control simply inserts a new DataPager control into its LayoutTemplate. The default paging markup generated by the ListView for the Grid layout is shown here:

```
<asp:datapager ID="DataPager1" runat="server">
    <Fields>
        <asp:nextpreviouspagerfield ButtonType="Button" FirstPageText="First"
            LastPageText="Last" NextPageText="Next" PreviousPageText="Previous"
            ShowFirstPageButton="True" ShowLastPageButton="True" />
    </Fields>
</asp:datapager>
```

The markup for the control shows that within the DataPager, a `Fields` collection has been created, which contains a `NextPreviousPagerField` object. As its name implies, using the `NextPreviousPager` object results in the DataPager rendering Next and Previous buttons as its user interface. The DataPager control includes three types of `Field` objects: the `NextPreviousPagerField`, the `NumericPagerField` object, which generates a simple numeric page list, and the `TemplatePagerField`, which allows you to specify your own custom paging user interface. Each of these different `Field` types includes a variety of properties that you can use to control exactly how the DataPager displays the user interface. Additionally, because the DataPager exposes a `Fields` collection rather than a simple `Field` property, you can actually display several different `Field` objects within a single DataPager control.

The TemplatePagerField is a unique type of `Field` object that contains no User Interface itself, but simply exposes a template that you can use to completely customize the pagers user interface. Listing 7-56 demonstrates the use of the TemplatePagerField.

Listing 7-56: Creating a custom DataPager user interface

```
<asp:DataPager ID="DataPager1" runat="server">
    <Fields>
        <asp:TemplatePagerField>
            <PagerTemplate>
                Page
                <asp:Label runat="server"
                    Text="<%# (Container.StartRowIndex/Container.PageSize)+1%>" />
                of
                <asp:Label runat="server"
                    Text="<%# Container.TotalRowCount/Container.PageSize%>" />
            </PagerTemplate>
        </asp:TemplatePagerField>
    </Fields>
</asp:DataPager>
```

Notice that the sample uses ASP.NET data binding to provide the total page count, page size and the row that the page should start on; these are values exposed by the DataPager control.

If you want to use custom navigation controls in the PagerTemplate, such as a Button control to change the currently display Page, you would create standard Click an event handler for the Button. Within that event handler you can access the DataPagers StartRowIndex, TotalRowCount and PageSize properties to calculate the new StartRowIndex the ListView should use when it renders.

Unlike the paging provided by the GridView, the DataPager control, because it is a separate control, gives you total freedom over where to place it on your webpage. The samples you have seem so far have all looked at the DataPager control when it is placed directly in a ListView, but the control can be placed anywhere on the webform.

Listing 7-57: Placing the DataPager control outside of the ListView

```
<asp:DataPager ID="DataPager1" runat="server" PagedControlID="ListView1">
    <Fields>
        <asp:NumericPagerField />
    </Fields>
</asp:DataPager>
```

In Listing 7-57, the only significant change you should notice is the the use of the PagedControlID property. This property allows you to specify explicitly which control this pager should work with.

FormView

The FormView control, introduced in the ASP.NET 2.0 toolbox, functions like the DetailsView control in that it displays a single data item from a bound data source control and allows adding, editing, and deleting data. What makes it unique is that it displays the data in custom templates, which gives you much greater control over how the data is displayed and edited. Figure 7-35 shows a FormView control ItemTemplate being edited in Visual Studio. You can see that you have complete control over how your data is displayed. The FormView control also contains an EditItemTemplate and InsertItemTemplate that allows you to determine how the control displays when entering edit or insert mode.

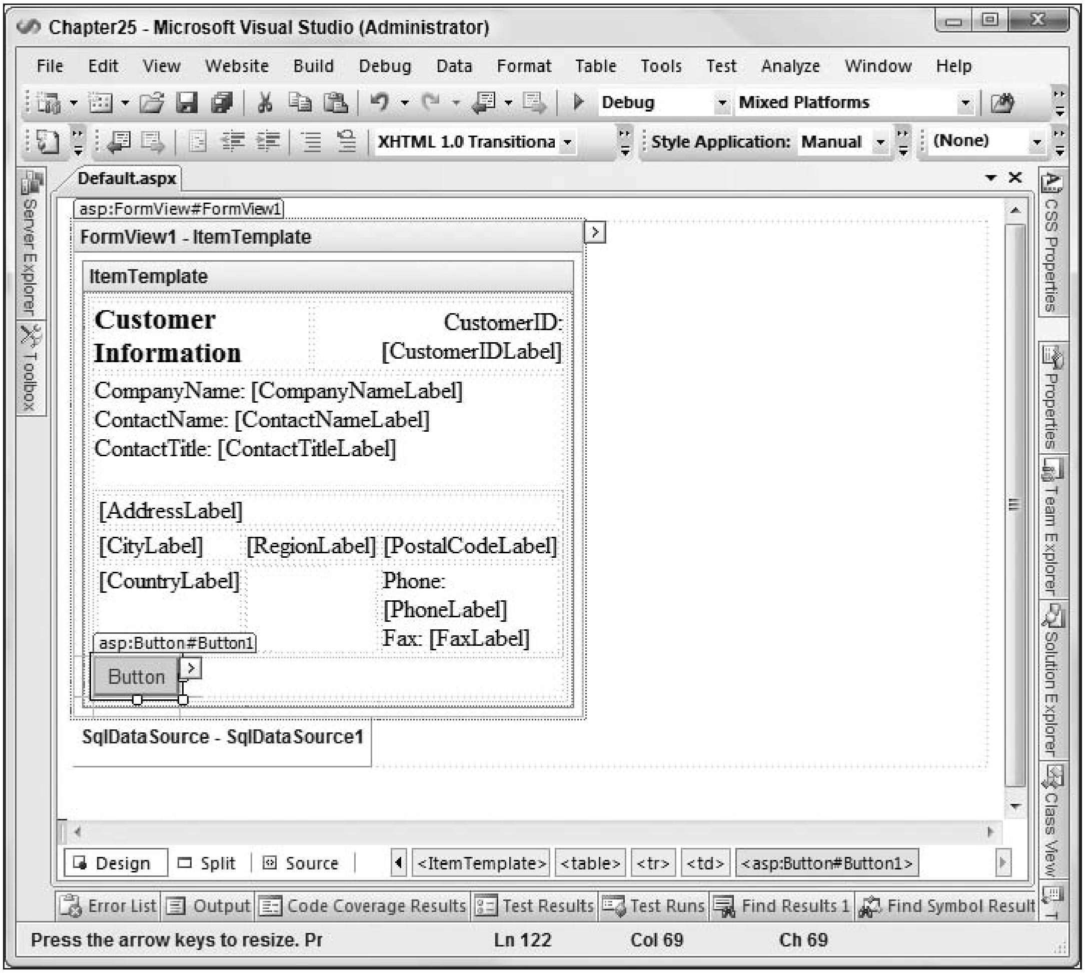
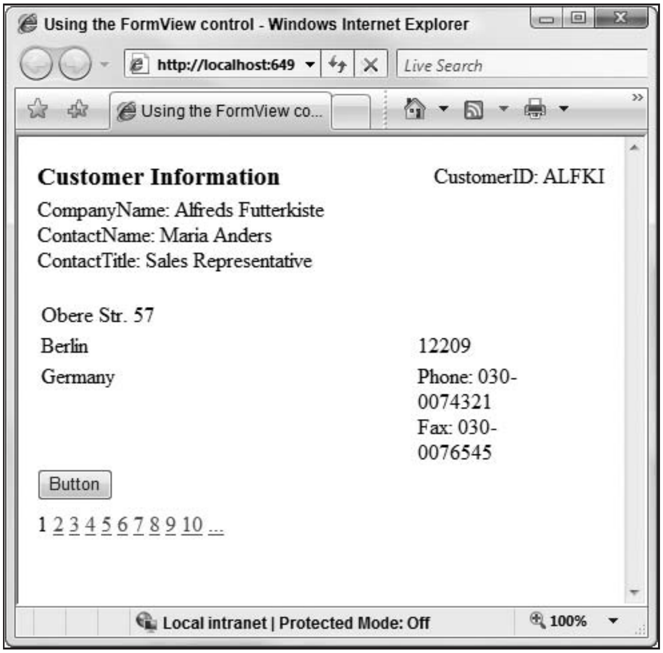

Chapter25 - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Data Format Table Tools Test Analyze Window Help
Debug
Mixed Platforms
XHTML 1.0 Transitiona
Style Application: Manual
(None)
Default.aspx
Server Explorer
Toolbox
CSS Properties
Properties
Team Explorer
Solution Explorer
Class View
asp:FormView#FormView1
FormView1 - ItemTemplate
ItemTemplate
Customer Information
CustomerID: [CustomerIDLabel]
CompanyName: [CompanyNameLabel]
ContactName: [ContactNameLabel]
ContactTitle: [ContactTitleLabel]
[AddressLabel]
[CityLabel]
[RegionLabel]
[PostalCodeLabel]
[CountryLabel]
Phone: [PhoneLabel]
Fax: [FaxLabel]
asp:Button#Button1
Button
SqlDataSource - SqlDataSource1
Design
Split
Source
<ItemTemplate> <table> <tr> <td> <asp:Button#Button1>
Error List
Output
Code Coverage Results
Test Results
Test Runs
Find Results 1
Find Symbol Result
Press the arrow keys to resize. Pr
Ln 122
Col 69
Ch 69

Figure 7-35

While Figure 7-35 shows the FormView control in action in Visual Studio, Figure 7-36 shows the control displaying its ItemTemplate, reflecting the custom layout that was designed in Visual Studio.

In Figure 7-37, you see the control in edit mode, showing the standard EditItemTemplate layout.

Listing 7-58 shows the code that Visual Studio generates when designing the FormView control's customized ItemTemplate.


Using the FormView control - Windows Internet Explorer
http://localhost:649
Live Search
Using the FormView co...
Local intranet | Protected Mode: Off
100%

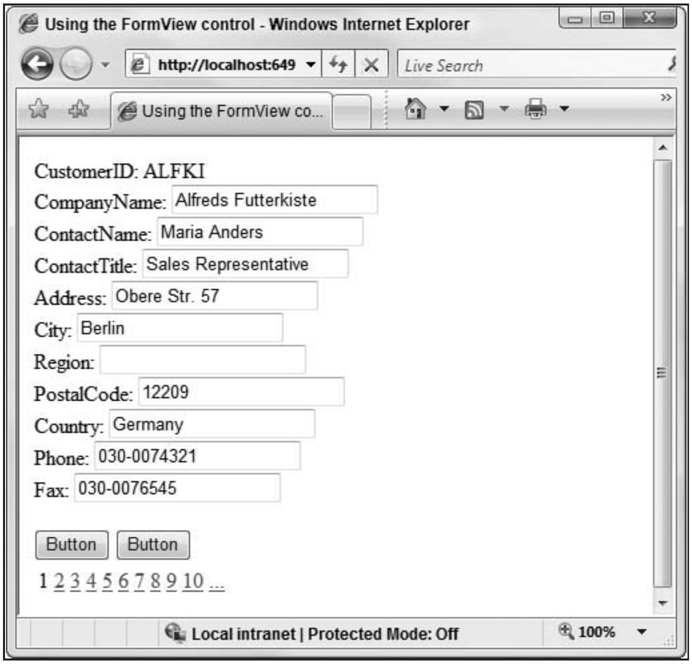
Customer Information CustomerID: ALFKI

CompanyName: Alfreds Futterkiste
ContactName: Maria Anders
ContactTitle: Sales Representative

Obere Str. 57
Berlin 12209
Germany Phone: 030-0074321
Fax: 030-0076545

Button
1 2 3 4 5 6 7 8 9 10 ...

Figure 7-36


Using the FormView control - Windows Internet Explorer
http://localhost:649
Live Search
Using the FormView co...
Local intranet | Protected Mode: Off
100%

CustomerID: ALFKI
CompanyName: Alfreds Futterkiste
ContactName: Maria Anders
ContactTitle: Sales Representative
Address: Obere Str. 57
City: Berlin
Region:
PostalCode: 12209
Country: Germany
Phone: 030-0074321
Fax: 030-0076545

Button Button
1 2 3 4 5 6 7 8 9 10 ...

Figure 7-37

Listing 7-58: Using a FormView control to display and edit data

```
<%@ Page Language="C#" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Using the FormView control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:FormView ID="FormView1" Runat="server" DataSourceID="SqlDataSource1"
             DataKeyNames="CustomerID" AllowPaging="True">
            <EditItemTemplate>
                CustomerID:
                <asp:Label Text='<%# Eval("CustomerID") %>' Runat="server"
                     ID="CustomerIDLabel1">
                </asp:Label><br />
                CompanyName:
                <asp:TextBox Text='<%# Bind("CompanyName") %>' Runat="server"
                     ID="CompanyNameTextBox"></asp:TextBox><br />
                ContactName:
                <asp:TextBox Text='<%# Bind("ContactName") %>' Runat="server"
                     ID="ContactNameTextBox"></asp:TextBox><br />
                ContactTitle:
                <asp:TextBox Text='<%# Bind("ContactTitle") %>' Runat="server"
                    ID="ContactTitleTextBox"></asp:TextBox><br />
                Address:
                <asp:TextBox Text='<%# Bind("Address") %>' Runat="server"
                     ID="AddressTextBox"></asp:TextBox><br />
                City:
                <asp:TextBox Text='<%# Bind("City") %>' Runat="server"
                     ID="CityTextBox"></asp:TextBox><br />
                Region:
                <asp:TextBox Text='<%# Bind("Region") %>' Runat="server"
                     ID="RegionTextBox"></asp:TextBox><br />
                PostalCode:
                <asp:TextBox Text='<%# Bind("PostalCode") %>' Runat="server"
                     ID="PostalCodeTextBox"></asp:TextBox><br />
                Country:
                <asp:TextBox Text='<%# Bind("Country") %>' Runat="server"
                     ID="CountryTextBox"></asp:TextBox><br />
                Phone:
                <asp:TextBox Text='<%# Bind("Phone") %>' Runat="server"
                     ID="PhoneTextBox"></asp:TextBox><br />
                Fax:
                <asp:TextBox Text='<%# Bind("Fax") %>' Runat="server"
                     ID="FaxTextBox"></asp:TextBox><br />
                <br />
                <asp:Button ID="Button2" Runat="server" Text="Button"
                     CommandName="update" />
                <asp:Button ID="Button3" Runat="server" Text="Button"
                     CommandName="cancel" />
```

Continued

```
                </EditItemTemplate>
                <ItemTemplate>
                    <table width="100%">
                        <tr>
                            <td style="width: 439px">
                            <b>
                            <span style="font-size: 14pt">Customer Information</span>
                            </b>
                            </td>
                            <td style="width: 439px" align="right">
                                CustomerID:
                                <asp:Label ID="CustomerIDLabel" Runat="server"
                                     Text='<%# Bind("CustomerID") %>'>
                                     </asp:Label></td>
                        </tr>
                        <tr>
                            <td colspan="2">
                                CompanyName:
                                <asp:Label ID="CompanyNameLabel" Runat="server"
                                     Text='<%# Bind("CompanyName") %>'>
                                     </asp:Label><br />
                                ContactName:
                                <asp:Label ID="ContactNameLabel" Runat="server"
                                     Text='<%# Bind("ContactName") %>'>
                                     </asp:Label><br />
                                ContactTitle:
                                <asp:Label ID="ContactTitleLabel" Runat="server"
                                     Text='<%# Bind("ContactTitle") %>'>
                                     </asp:Label><br />
                                <br />
                                <table width="100%"><tr>
                                        <td colspan="3">
                                            <asp:Label ID="AddressLabel" Runat="server"
                                                 Text='<%# Bind("Address") %>'>
                                                 </asp:Label></td>
                                    </tr>
                                    <tr>
                                        <td style="width: 100px">
                                            <asp:Label ID="CityLabel" Runat="server"
                                                 Text='<%# Bind("City") %>'>
                                                 </asp:Label></td>
                                        <td style="width: 100px">
                                            <asp:Label ID="RegionLabel" Runat="server"
                                                 Text='<%# Bind("Region") %>'>
                                                 </asp:Label></td>
                                        <td style="width: 100px">
                                            <asp:Label ID="PostalCodeLabel"
                                                Runat="server"
                                                Text='<%# Bind("PostalCode") %>'>
                                                </asp:Label>
                                        </td>
                                    </tr>
                                    <tr>
                                        <td style="width: 100px" valign="top">
```

Continued

```
                                        <asp:Label ID="CountryLabel" Runat="server"
                                             Text='<%# Bind("Country") %>'>
                                             </asp:Label></td>
                                    <td style="width: 100px"></td>
                                    <td style="width: 100px">
                                        Phone:
                                        <asp:Label ID="PhoneLabel" Runat="server"
                                                   Text='<%# Bind("Phone") %>'>
                                                   </asp:Label><br />
                                        Fax:
                                        <asp:Label ID="FaxLabel" Runat="server"
                                             Text='<%# Bind("Fax") %>'>
                                             </asp:Label><br />
                                    </td>
                                </tr></table>
                            <asp:Button ID="Button1" Runat="server"
                                 Text="Button" CommandName="edit" />
                        </td>
                    </tr></table>
            </ItemTemplate>
        </asp:FormView>
        <asp:SqlDataSource ID="SqlDataSource1" Runat="server"
             SelectCommand="SELECT * FROM [Customers]"
             ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>">
        </asp:SqlDataSource>

    </div>
    </form>
</body>
</html>
```

Other Databound Controls

ASP.NET 1.0/1.1 contained many other controls that could be bound to data sources. ASP.NET 2.0 retained those controls, enhanced some, and added several additional bound controls to the toolbox ASP.NET 3.5 continues to include these controls in its toolbox.

DropDownList, ListBox, RadioButtonList, and CheckBoxList

Although the DropDownList, ListBox, and CheckBoxList controls largely remained the same from ASP.NET 1.0/1.1 to ASP.NET 2.0, several new properties that you might find useful were added. In addition, ASP.NET 2.0 added new RadioButtonList and BulletedList controls. All of these controls remain the same in ASP.NET 3.5.

One of the new properties available in all these controls is the `AppendDataBoundItems` property. Setting this property to `True` tells the DropDownList control to append data-bound list items to any existing statically declared items, rather then overwriting them as the ASP.NET 1.0/1.1 version would have done.

Another useful new property available to all these controls is the `DataTextFormatString`, which allows you to specify a string format for the display text of the DropDownList items.

TreeView

Another control included in the ASP.NET toolbox is the TreeView control. Because the TreeView can display only hierarchical data, it can be bound only to the XmlDataSource and the SiteMapDataSource controls. Listing 7-59 shows a sample SiteMap file you can use for your SiteMapDataSource control.

Listing 7-59: A SiteMap file for your samples

```
<siteMap>
    <siteMapNode url="page3.aspx" title="Home" description="" roles="">
        <siteMapNode url="page2.aspx" title="Content" description="" roles="" />
        <siteMapNode url="page4.aspx" title="Links" description="" roles="" />
        <siteMapNode url="page1.aspx" title="Comments" description="" roles="" />
    </siteMapNode>
</siteMap>
```

Listing 7-60 shows how you can bind a TreeView control to a SiteMapDataSource control to generate navigation for your Web site.

Listing 7-60: Using the TreeView with a SqlDataSource control

```
<%@ Page Language="C#" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Using the TreeView control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TreeView ID="TreeView1" Runat="server"
             DataSourceID="SiteMapDataSource1">
        </asp:TreeView>
        <asp:SiteMapDataSource ID="SiteMapDataSource1" Runat="server" />

    </div>
    </form>
</body>
</html>
```

Ad Rotator

The familiar AdRotator control was greatly enhanced in ASP.NET 2.0. You can see the control by using the SqlDataSource or XmlDataSource controls. Listing 7-61 shows an example of binding the AdRotator to a SqlDataSource control.

Listing 7-61: Using the AdRotator with a SqlDataSource control

```
<asp:AdRotator ID="AdRotator1" runat="server"
     DataSourceId="SqlDataSource1" AlternateTextField="AlternateTF"
     ImageUrlField="Image" NavigateUrlField="NavigateUrl" />
```

For more information on the Ad Rotator control, see Chapter 5.

Menu

The last control in this section is the new Menu control. Like the TreeView control, it is capable of displaying hierarchical data in a vertical *pop-out style* menu. Also like the TreeView control, it can be bound only to the XmlDataSource and the SiteMapDataSource controls. Listing 7-62 shows how you can use the same SiteMap data used earlier in the TreeView control sample, and modify it to display using the new Menu control.

Listing 7-62: Using the Menu control with a SiteMap

```
<%@ Page Language="C#" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Using the Menu control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Menu ID="Menu1" Runat="server" DataSourceID="SiteMapDataSource1">
        </asp:Menu>
        <asp:SiteMapDataSource ID="SiteMapDataSource1" Runat="server" />
    </div>
    </form>
</body>
</html>
```

For more information on using the Menu control, see Chapter 14.

Inline Data-Binding Syntax

Another feature of data binding in ASP.NET is inline data-binding syntax. Inline syntax in ASP.NET 1.0/1.1 was primarily relegated to templated controls such as the DataList or the Repeater controls, and even then it was sometimes difficult and confusing to make it work as you wanted it to. In ASP.NET 1.0/1.1, if you needed to use inline data binding, you might have created something like the procedure shown in Listing 7-63.

Listing 7-63: Using DataBinders in ASP.NET 1.0

```
<asp:Repeater id=Repeater1 runat="server">
    <HeaderTemplate>
        <table>
    </HeaderTemplate>
    <ItemTemplate>
       <tr>
          <td>
              <%# Container.DataItem("Name") %><BR/>
              <%# Container.DataItem("Department") %><BR/>
              <%# DataBinder.Eval(
                     Container.DataItem, "HireDate", "{0:mm dd yyyy}") %><BR/>
           </td>
```

Continued

```
            </tr>
        </ItemTemplate>
        <FooterTemplate>
            </table>
        </FooterTemplate>
    </asp:Repeater>
```

As you can see in this sample, you are using a Repeater control to display a series of employees. Because the Repeater control is a templated control, you use data binding to output the employee-specific data in the proper location of the template. Using the `Eval` method also allows you to provide formatting information such as Date or Currency formatting at render-time.

In later versions of ASP.NET, including 3.5, the concept of inline data binding remains basically the same, but you are given a simpler syntax and several powerful new binding tools to use.

Data-Binding Syntax Changes

ASP.NET contains three different ways to perform data binding. First, you can continue to use the existing method of binding, using the `Container.DataItem` syntax:

```
<%# Container.DataItem("Name") %>
```

This is good because it means you won't have to change your existing Web pages if you are migrating from prior versions of ASP.NET. But if you are creating new Web pages, you should probably use the simplest form of binding, using the `Eval` method directly:

```
<%# Eval("Name") %>
```

You can also continue to format data using the formatter overload of the `Eval` method:

```
<%# Eval("HireDate", "{0:mm dd yyyy}" ) %>
```

In addition to these changes, ASP.NET includes a form of data binding called *two-way data binding*. In ASP.NET 1.0/1.1, using the data-binding syntax was essentially a read-only form of accessing data. Since the introduction of ASP.NET 2.0, two-way data binding has allowed you to support both read and write operations for bound data. This is done using the `Bind` method, which, other than using a different method name, works just like the `Eval` method:

```
<%# Bind("Name") %>
```

The new `Bind` method should be used in new controls such as the GridView, DetailsView, or FormView, where auto-updates to the data source are implemented.

When working with the data binding statements, remember that anything between the `<%# %>` delimiters is treated as an expression. This is important because it gives you additional functionality when data binding. For example, you could append additional data:

```
<%# "Foo " + Eval("Name") %>
```

Or you can even pass the evaluated value to a method:

```
<%# DoSomeProcess( Eval("Name") )%>
```

XML Data Binding

Because XML is becoming ever more prevalent in applications, ASP.NET also includes several ways to bind specifically to XML data sources. These new data binding expressions give you powerful ways of working with the hierarchical format of XML. Additionally, except for the different method names, these binding methods work exactly the same as the `Eval` and `Bind` methods discussed earlier. These binders should be used when you are using the XmlDataSource control. The first binding format that uses the `XPathBinder` class is shown in the following code:

```
<% XPathBinder.Eval(Container.DataItem, "employees/employee/Name") %>
```

Notice that rather than specifying a column name as in the `Eval` method, the `XPathBinder` binds the result of an XPath query. Like the standard `Eval` expression, the XPath data binding expression also has a shorthand format:

```
<% XPath("employees/employee/Name") %>
```

Also, like the `Eval` method, the XPath data binding expression supports applying formatting to the data:

```
<% XPath("employees/employee/HireDate", "{0:mm dd yyyy}") %>
```

The `XPathBinder` returns a single node using the XPath query provided. Should you want to return multiple nodes from the XmlDataSource Control, you can use the class's `Select` method. This method returns a list of nodes that match the supplied XPath query:

```
<% XPathBinder.Select(Container.DataItem,"employees/employee") %>
```

Or use the shorthand syntax:

```
<% XpathSelect("employees/employee") %>
```

Expressions and Expression Builders

Finally, ASP.NET introduces the concept of expressions and expression builders. Expressions are statements that are parsed by ASP.NET at runtime in order to return a data value. ASP.NET automatically uses expressions to do things like retrieve the database connection string when it parses the SqlDataSource control, so you may have already seen these statements in your pages. An example of the connection string Expression is shown in Listing 7-64.

Listing 7-64: A connection string Expression

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
     ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
    SelectCommand="SELECT * FROM Customers"></asp:SqlDataSource>
```

When ASP.NET is attempting to parse an ASP.NET Web page, it looks for expressions contained in the `<%$ %>`delimiters. This indicates to ASP.NET that this is an expression to be parsed. As shown in the previous listing, it attempts to locate the `NorthwindConnectionString` value from the web.config file. ASP.NET knows to do this because of the `ConnectionStrings` expression prefix, which tells ASP.NET to use the `ConnectionStringsExpressionBuilder` class to parse the expression.

ASP.NET includes several expression builders, including one for retrieving values from the AppSettings section of `web.config`: one for retrieving ConnectionStrings as shown in Listing 7-64, and one for retrieving localized resource file values. Listings 7-65 and 7-66 demonstrate using the `AppSettingsExpressionBuilder` and the `ResourceExpressionBuilder`.

Listing 7-65: Using AppSettingsExpressionBuilder

```
<asp:Label runat="server" ID="lblAppSettings"
    Text="<%$ AppSettings: LabelText %>"></asp:Label>
```

Listing 7-66: Using ResourceExpressionBuilder

```
<asp:Label runat="server" ID="lblResource"
    Text="<%$ Resources: TEST %>"></asp:Label>
```

In addition to using the expression builder classes, you can also create your own expressions by deriving a class from the `System.Web.Compilation.ExpressionBuilder` base class. This base class provides you with the overridable methods you need to implement if you want ASP.NET to parse your expression properly. Listing 7-67 shows a simple custom expression builder.

Listing 7-67: Using a simple custom expression builder

VB

```
Imports System;
Imports System.CodeDom;
Imports System.Web.Compilation;
Imports System.Web.UI;

<ExpressionPrefix("MyCustomExpression")>
<ExpressionEditor("MyCustomExpressionEditor")>
Public Class MyCustomExpression Inherits ExpressionBuilder

        Public Overrides Function
        GetCodeExpression(ByVal entry As BoundPropertyEntry,
            ByVal parsedData As object, ByVal context As ExpressionBuilderContext)
            As System.CodeDom.CodeExpression

        Return New CodeCastExpression("Int64", new CodePrimitiveExpression(1000))
    End Function
End Class
```

C#

```
using System;
using System.CodeDom;
using System.Web.Compilation;
using System.Web.UI;

[ExpressionPrefix("MyCustomExpression")]
[ExpressionEditor("MyCustomExpressionEditor")]
public class MyCustomExpression : ExpressionBuilder
{
    public override System.CodeDom.CodeExpression
```

```
            GetCodeExpression(BoundPropertyEntry entry, object parsedData,
                ExpressionBuilderContext context)
        {
            return new CodeCastExpression("Int64", new CodePrimitiveExpression(1000));
        }
    }
```

In examining this sample, notice several items. First, you have derived the `MyCustomExpression` class from `ExpressionBuilder` as I discussed earlier. Second, you have overridden the `GetCodeExpression` method. This method supplies you with several parameters that can be helpful in executing this method, and it returns a `CodeExpression` object to ASP.NET that it can execute at runtime to retrieve the data value.

> *The* `CodeExpression` *class is a base class in .NET's CodeDom infrastructure. Classes that are derived from the* `CodeExpression` *class provide abstracted ways of generating .NET code, whether VB or C#. This CodeDom infrastructure is what helps you create and run code dynamically at runtime.*

The `BoundPropertyEntry` parameter entry tells you exactly which property the expression is bound to. For example, in Listings 7-58 and 7-59, the Label's `Text` property is bound to the `AppSettings` and `Resources` expressions. The object parameter `parsedData` contains any data that was parsed and returned by the `ParseExpression` method that you see later on in the chapter. Finally, the `ExpressionBuilderContext` parameter context allows you to reference the virtual path or templated control associated with the expression.

In the body of the `GetCodeExpression` method, you are creating a new `CodeCastExpression` object, which is a class derived from the `CodeExpression` base class. The `CodeCastExpression` tells .NET to generate the appropriate code to execute a cast from one data type to another. In this case, you are casting the value 1000 to an Int64 datatype. When .NET executes the `CodeCastExpression`, it is (in a sense) writing the C# code ((long)(1000)), or (if your application was written in VB) CType(1000,Long). Note that a wide variety of classes derive from the `CodeExpression` class that you can use to generate your final code expression.

The final lines to note are the two attributes that have been added to the class. The `ExpressionPrefix`and `ExpressionEditor` attributes help .NET figure out that this class should be used as an expression, and they also help .NET locate the proper expression builder class when it comes time to parse the expression.

After you have created your expression builder class, you let .NET know about it. You do this by adding an `expressionBuilders`node to the compilation node in your `web.config` file. Notice that the value of the `ExpressionPrefix` is added to the `expressionBuilder` to help ASP.NET locate the appropriate expression builder class at runtime.

```
<compilation debug="true" strict="false" explicit="true">
    <expressionBuilders>
        <add expressionPrefix="MyCustomExpression" type="MyCustomExpression"/>
    </expressionBuilders>
</compilation>
```

The `GetCodeExpression` method is not the only member available for overriding in the `ExpressionBuilder` class. Several other useful members include the `ParseExpression`, `SupportsEvaluate`, and `EvaluateExpression` methods.

The `ParseExpression` method lets you pass parsed expression data into the `GetCodeExpression` method. For example, in Listing 7-60, the `CodeCastExpression` value 1000 was hard-coded. If, however, you want to allow a developer to pass that value in as part of the expression, you simply use the `ParseExpression` method as shown in Listing 7-68

Listing 7-68: Using ParseExpression

VB

```
Imports System;
Imports System.CodeDom;
Imports System.Web.Compilation;
Imports System.Web.UI;

<ExpressionPrefix("MyCustomExpression")>
<ExpressionEditor("MyCustomExpressionEditor")>
Public Class MyCustomExpression Inherits ExpressionBuilder

    Public Overrides Function
        GetCodeExpression(ByVal entry As BoundPropertyEntry,
            ByVal parsedData As object, ByVal context As ExpressionBuilderContext)
            As System.CodeDom.CodeExpression

         Return New CodeCastExpression("Int64",
             new CodePrimitiveExpression(passedData))
    End Function

     Public Overrides Function ParseExpression
         (ByVal expression As String, ByVal propertyType As Type,
             ByVal context As ExpressionBuilderContext) As Object

          Return expression
     End Function
End Class
```

C#

```
using System;
using System.CodeDom;
using System.Web.Compilation;
using System.Web.UI;

[ExpressionPrefix("MyCustomExpression")]
[ExpressionEditor("MyCustomExpressionEditor")]
public class MyCustomExpression : ExpressionBuilder
{
    public override System.CodeDom.CodeExpression
        GetCodeExpression(BoundPropertyEntry entry, object parsedData,
            ExpressionBuilderContext context)
    {
         return new CodeCastExpression("Int64",
             new CodePrimitiveExpression(parsedData));
    }
```

```
      public override object ParseExpression
          (string expression, Type propertyType, ExpressionBuilderContext context)
      {
           return expression;
      }
  }
```

The last two `ExpressionBuilder` overrides to examine are the `SupportsEvaluate` and `EvaluateExpression` members. You need to override these methods if you are running your Web site in a no-compile scenario (you have specified `compilationMode = "never"` in your @Page directive). The `SupportEvaluate` property returns a Boolean indicating to ASP.NET whether this expression can be evaluated while a page is executing in no-compile mode. If `True` is returned and the page is executing in no-compile mode, the `EvaluateExpression` method is used to return the data value rather than the `GetCodeExpression` method. The `EvaluateExpression` returns an object representing the data value. See Listing 7-69.

Listing 7-69: Overriding SupportsEvaluate and EvaluateExpression

VB

```
Imports System;
Imports System.CodeDom;
Imports System.Web.Compilation;
Imports System.Web.UI;

<ExpressionPrefix("MyCustomExpression")>
<ExpressionEditor("MyCustomExpressionEditor")>
Public Class MyCustomExpression Inherits ExpressionBuilder

    Public Overrides Function
        GetCodeExpression(ByVal entry As BoundPropertyEntry,
            ByVal parsedData As object, ByVal context As ExpressionBuilderContext)
            As System.CodeDom.CodeExpression

        Return New CodeCastExpression("Int64",
            new CodePrimitiveExpression(pasedData))
    End Function

    Public Overrides Function ParseExpression
        (ByVal expression As String, ByVal propertyType As Type,
            ByVal context As ExpressionBuilderContext) As Object

         Return expression
    End Function

    Public Overrides ReadOnly Property SupportsEvaluate as Boolean
        Get
            Return True
        End Get
    End Property

    Public Overrides Function EvaluateExpression(ByVal target As Object,
        ByVal Entry As BoundPropertyEntry, ByVal parsedData As Object,
        ByVal context As ExpressionBuilderContext) as Object
```

Continued

```
        Return parsedData;
    End Function

End Class
```

C#

```
using System;
using System.CodeDom;
using System.Web.Compilation;
using System.Web.UI;

[ExpressionPrefix("MyCustomExpression")]
[ExpressionEditor("MyCustomExpression123Editor")]
public class MyCustomExpression : ExpressionBuilder
{
    public override System.CodeDom.CodeExpression
        GetCodeExpression(BoundPropertyEntry entry, object parsedData,
            ExpressionBuilderContext context)
    {
        return new CodeCastExpression("Int64",
            new CodePrimitiveExpression(parsedData));
    }

    public override object ParseExpression
        (string expression, Type propertyType, ExpressionBuilderContext context)
    {
         return expression;
    }

    public override bool SupportsEvaluate
    {
        get
        {
            return true;
        }
    }

    public override object EvaluateExpression(object target,
        BoundPropertyEntry entry, object parsedData,
        ExpressionBuilderContext context)
    {
        return parsedData;
    }
}
```

As shown in Listing 7-69, you can simply return `True` from the `SupportsEvaluate` property if you want to override the `EvaluateExpression` method. Then all you do is return an object from the `EvaluateExpression` method.

Summary

In this chapter, you examined data binding in ASP.NET. The introduction of data source controls such as the LinqDataSource control, SqlDataSource control, or the XmlDataSource control makes querying and displaying data from any number of data sources an almost trivial task. Using the data source controls' own wizards, you learned how easy it is to generate powerful data access functionality with almost no code required.

You examined how even a beginning developer can easily combine the data source controls with the GridView, ListView, and DetailsView controls to create powerful data manipulation applications with a minimal amount of coding.

You saw how ASP.NET includes a multitude of controls that can be data-bound, specifically examining how, since ASP.NET 1.0/1.1, many controls have been enhanced, and examining the features of the data bound controls that are included in the ASP.NET toolbox, such as the GridView, TreeView, ListView, and Menu controls, the new FormView control.

Finally, you looked at how the inline data-binding syntax has been improved and strengthened with the addition of the XML-specific data-binding expressions.

8

Data Management with ADO.NET

This chapter provides information on programming with the data management features that are part of ADO.NET, a key component of the .NET Framework and of your ASP.NET development. The discussion begins with the basics of ADO.NET and later dives into the ways you can use various features that make up ADO.NET to manage data contained in a relational database.

ADO.NET, first introduced in version 1.0 of the .NET Framework, provided an extensive array of features to handle live data in a connected mode or data that is disconnected from its underlying data store. ADO.NET 1.0 was primarily developed to address two specific problems in getting at data. The first had to do with the user's need to access data once and to iterate through a collection of data in a single instance. This need often arose in Web application development.

ADO.NET addresses a couple of the most common data-access strategies that are used for applications today. When classic ADO was developed, many applications could be connected to the data store almost indefinitely. Today, with the explosion of the Internet as the means of data communication, a new data technology is required to make data accessible and updateable in a disconnected architecture.

The first of these common data-access scenarios is one in which a user must locate a collection of data and iterate through this data a single time. This is a popular scenario for Web pages. When a request for data from a Web page that you have created is received, you can simply fill a table with data from a data store. In this case, you go to the data store, grab the data that you want, send the data across the wire, and then populate the table. In this scenario, the goal is to get the data in place as fast as possible.

The second way to work with data in this disconnected architecture is to grab a collection of data and use this data separately from the data store itself. This could be on the server or even on the client. Although the data is disconnected, you want the ability to keep the data (with all of its tables and relations in place) on the client side. Classic ADO data was represented by a single table that you could iterate through. ADO.NET, however, can be a reflection of the data store itself, with tables, columns, rows, and relations all in place. When you are done with the client-side copy of the data, you can persist the changes that you made in the local copy of data directly back into the data store. The technology that gives you this capability is the DataSet, which will be covered shortly.

Although classic ADO was geared for a two-tiered environment (client-server), ADO.NET addresses a multi-tiered environment. ADO.NET is easy to work with because it has a unified programming model. This unified programming model makes working with data on the server the same as working with data on the client. Because the models are the same, you find yourself more productive when working with ADO.NET.

Basic ADO.NET Features

This chapter begins with a quick look at the basics of ADO.NET and then provides an overview of basic ADO.NET capabilities, namespaces, and classes. It also reviews how to work with the `Connection`, `Command`, `DataAdapter`, `DataSet`, and `DataReader` objects.

Common ADO.NET Tasks

Before jumping into the depths of ADO.NET, step back and make sure that you understand some of the common tasks you might perform programmatically within ADO.NET. This next section looks at the process of selecting, inserting, updating, and deleting data.

> *The following example makes use of the* `Northwind.mdf` *SQL Server Express Database file. To get this database, please search for ''Northwind and pubs Sample Databases for SQL Server 2000''. You can find this link at* `www.microsoft.com/downloads/details.aspx?familyid=06616212-0356-46a0-8da2-eebc53a68034&displaylang=en`*. Once installed, you will find the Northwind.mdf file in the C:\ SQL Server 2000 Sample Databases directory. To add this database to your ASP.NET application, create an* `App_Data` *folder within your project (if it isn't already there) and right-click on the folder and select Add Existing Item. From the provided dialog box, you are then able to browse to the location of the* `Northwind.mdf` *file that you just installed. If you are having trouble getting permissions to work with the database, make a data connection to the file from the Visual Studio Server Explorer by right-clicking on the Data Connections node and selecting Add New Connection from the provided menu. You will be asked to be made the appropriate user of the database. ThenVS will make the appropriate changes on your behalf for this to occur.*

Selecting Data

After the connection to the data source is open and ready to use, you probably want to read the data from the data source. If you do not want to manipulate the data, but simply to read it or transfer it from one spot to another, you use the `DataReader` class.

In the following example (Listing 8-1), you use the `GetCompanyNameData()` function to provide a list of company names from the SQL Northwind database.

Listing 8-1: Reading the data from a SQL database using the DataReader class

VB

```
Imports Microsoft.VisualBasic
Imports System.Collections.Generic
```

Continued

```
Imports System.Data
Imports System.Data.SqlClient

Public Class SelectingData
    Public Function GetCompanyNameData() As List(Of String)
        Dim conn As SqlConnection
        Dim cmd As SqlCommand
        Dim cmdString As String = "Select CompanyName from Customers"
        conn = New SqlConnection("Data Source=.\SQLEXPRESS;AttachDbFilename=
           |DataDirectory|\NORTHWND.MDF;Integrated Security=True;
           User Instance=True") ' Put this string on one line in your code
        cmd = New SqlCommand(cmdString, conn)
        conn.Open()

        Dim myReader As SqlDataReader
        Dim returnData As List(Of String) = New List(Of String)
        myReader = cmd.ExecuteReader(CommandBehavior.CloseConnection)

        While myReader.Read()
            returnData.Add(myReader("CompanyName").ToString())
        End While

        Return returnData
    End Function
End Class
```

C#

```
using System.Data;
using System.Data.SqlClient;
using System.Collections.Generic;

public class SelectingData
{
    public List<string> GetCompanyNameData()
    {
        SqlConnection conn;
        SqlCommand cmd;
        string cmdString = "Select CompanyName from Customers";
        conn = new
            SqlConnection(@"Data Source=.\SQLEXPRESS;AttachDbFilename=
                |DataDirectory|\NORTHWND.MDF;Integrated Security=True;
                User Instance=True");  // Put this string on one line in your code
        cmd = new SqlCommand(cmdString, conn);
        conn.Open();

        SqlDataReader myReader;
        List<string> returnData = new List<string>();

        myReader = cmd.ExecuteReader(CommandBehavior.CloseConnection);

        while (myReader.Read())
        {
            returnData.Add(myReader["CompanyName"].ToString());
```

Continued

```
        }

        return returnData;
    }
}
```

In this example, you create an instance of both the `SqlConnection` and the `SqlCommand` classes. Then, before you open the connection, you simply pass the `SqlCommand` class a SQL command selecting specific data from the Northwind database. After your connection is opened (based upon the commands passed in), you create a `DataReader`. To read the data from the database, you iterate through the data with the `DataReader` by using the `myReader.Read()` method. After the `List(Of String)` object is built, the connection is closed, and the object is returned from the function.

Inserting Data

When working with data, you often insert the data into the data source. Listing 8-2 shows you how to do this. This data may have been passed to you by the end user through the XML Web Service, or it may be data that you generated within the logic of your class.

Listing 8-2: Inserting data into SQL Server

VB

```
Public Sub InsertData()
    Dim conn As SqlConnection
    Dim cmd As SqlCommand
    Dim cmdString As String = "Insert Customers (CustomerID, _
       CompanyName, ContactName) Values ('BILLE', 'XYZ Company', 'Bill Evjen')"
    conn = New SqlConnection("Data Source=.\SQLEXPRESS;AttachDbFilename=
           |DataDirectory|\NORTHWND.MDF;Integrated Security=True;
           User Instance=True") ' Put this string on one line in your code
    cmd = New SqlCommand(cmdString, conn)
    conn.Open()

    cmd.ExecuteNonQuery()
    conn.Close()
End Sub
```

C#

```
public void InsertData()
{
    SqlConnection conn;
    SqlCommand cmd;
    string cmdString = "Insert Customers (CustomerID, CompanyName,
       ContactName) Values ('BILLE', 'XYZ Company', 'Bill Evjen')";
    conn = new
           SqlConnection(@"Data Source=.\SQLEXPRESS;AttachDbFilename=
               |DataDirectory|\NORTHWND.MDF;Integrated Security=True;
               User Instance=True");  // Put this string on one line in your code
    cmd = new SqlCommand(cmdString, conn);
    conn.Open();

    cmd.ExecuteNonQuery();
    conn.Close();
}
```

Inserting data into SQL is pretty straightforward and simple. Using the SQL command string, you insert specific values for specific columns. The actual insertion is initiated using the `cmd.ExecuteNonQuery()` command. This executes a command on the data when you don't want anything in return.

Updating Data

In addition to inserting new records into a database, you frequently update existing rows of data in a table. Imagine a table in which you can update multiple records at once. In the example in Listing 8-3, you want to update an employee table by putting a particular value in the `emp_bonus` column if the employee has been at the company for five years or longer.

Listing 8-3: Updating data in SQL Server

VB

```
Public Function UpdateEmployeeBonus() As Integer
    Dim conn As SqlConnection
    Dim cmd As SqlCommand
    Dim RecordsAffected as Integer
    Dim cmdString As String = "UPDATE Employees SET emp_bonus=1000 WHERE " & _
       "yrs_duty>=5"
    conn = New SqlConnection("Data Source=.\SQLEXPRESS;AttachDbFilename=
           |DataDirectory|\NORTHWND.MDF;Integrated Security=True;
           User Instance=True") ' Put this string on one line in your code
    cmd = New SqlCommand(cmdString, conn)
    conn.Open()

    RecordsAffected = cmd.ExecuteNonQuery()
    conn.Close()

    Return RecordsAffected
End Function
```

C#

```
public int UpdateEmployeeBonus()
{
    SqlConnection conn;
    SqlCommand cmd;
    int RecordsAffected;
    string cmdString = "UPDATE Employees SET emp_bonus=1000 WHERE yrs_duty>=5";
    conn = new
        SqlConnection(@"Data Source=.\SQLEXPRESS;AttachDbFilename=
            |DataDirectory|\NORTHWND.MDF;Integrated Security=True;
            User Instance=True");  // Put this string on one line in your code

    cmd = new SqlCommand(cmdString, conn);
    conn.Open();

    RecordsAffected = cmd.ExecuteNonQuery();
    conn.Close();

    return RecordsAffected;
}
```

This update function iterates through all the employees in the table and changes the value of the emp_bonus field to `1000` if an employee has been with the company for more than five years. This is done with the SQL command string. The great thing about these update capabilities is that you can capture the number of records that were updated by assigning the `ExecuteNonQuery()` command to the `RecordsAffected` variable. The total number of affected records is then returned by the function.

Deleting Data

Along with reading, inserting, and updating data, you sometimes need to delete data from the data source. Deleting data is a simple process of using the SQL command string and then the `ExecuteNonQuery()` command as you did in the update example. See Listing 8-4 for an illustration of this.

Listing 8-4: Deleting data from SQL Server

VB

```
Public Function DeleteEmployee() As Integer
    Dim conn As SqlConnection
    Dim cmd As SqlCommand
    Dim RecordsAffected as Integer
    Dim cmdString As String = "DELETE Employees WHERE LastName='Evjen'"
    conn = New SqlConnection("Data Source=.\SQLEXPRESS;AttachDbFilename=
           |DataDirectory|\NORTHWND.MDF;Integrated Security=True;
           User Instance=True") ' Put this string on one line in your code
    cmd = New SqlCommand(cmdString, conn)
    conn.Open()

    RecordsAffected = cmd.ExecuteNonQuery()
    conn.Close()

    Return RecordsAffected
End Function
```

C#

```
public int DeleteEmployee()
{
    SqlConnection conn;
    SqlCommand cmd;
    int RecordsAffected;
    string cmdString = "DELETE Employees WHERE LastName='Evjen'";
    conn = new
        SqlConnection(@"Data Source=.\SQLEXPRESS;AttachDbFilename=
            |DataDirectory|\NORTHWND.MDF;Integrated Security=True;
            User Instance=True");  // Put this string on one line in your code
    cmd = new SqlCommand(cmdString, conn);
    conn.Open();

    RecordsAffected = cmd.ExecuteNonQuery();
    conn.Close();

    return RecordsAffected;
}
```

You can assign the `ExecuteNonQuery()` command to an `Integer` variable (just as you did for the update function) to return the number of records deleted.

Basic ADO.NET Namespaces and Classes

The six core ADO.NET namespaces are shown in the following table. In addition to these namespaces, each new data provider can have its own namespace. As an example, the Oracle .NET data provider adds a namespace of `System.Data.OracleClient` (for the Microsoft-built Oracle data provider).

Namespace	Description
`System.Data`	This namespace is the core of ADO.NET. It contains classes used by all data providers. It contains classes to represent tables, columns, rows, and the `DataSet` class. It also contains several useful interfaces, such as `IDbCommand`, `IDbConnection`, and `IDbDataAdapter`. These interfaces are used by all managed providers, enabling them to plug into the core of ADO.NET.
`System.Data.Common`	This namespace defines common classes that are used as base classes for data providers. All data providers share these classes. A few examples are `DbConnection` and `DbDataAdapter`.
`System.Data.OleDb`	This namespace defines classes that work with OLE-DB data sources using the .NET OleDb data provider. It contains classes such as `OleDbConnection` and `OleDbCommand`.
`System.Data.Odbc`	This namespace defines classes that work with the ODBC data sources using the .NET ODBC data provider. It contains classes such as `OdbcConnection` and `OdbcCommand`.
`System.Data.SqlClient`	This namespace defines a data provider for the SQL Server 7.0 or higher database. It contains classes such as `SqlConnection` and `SqlCommand`.
`System.Data.SqlTypes`	This namespace defines a few classes that represent specific data types for the SQL Server database.

ADO.NET has three distinct types of classes commonly referred to as `Disconnected`, `Shared`, and `Data Providers`. The `Disconnected` classes provide the basic structure for the ADO.NET framework. A good example of this type of class is the `DataTable` class. The objects of this class are capable of storing data without any dependency on a specific data provider. The `Shared` classes form the base classes for data providers and are shared among all data providers. The `Data Provider` classes are meant to work with different kinds of data sources. They are used to perform all data-management operations on specific databases. The `SqlClient` data provider, for example, works only with the SQL Server database.

A data provider contains `Connection`, `Command`, `DataAdapter`, and `DataReader` objects. Typically, in programming ADO.NET, you first create the `Connection` object and provide it with the necessary information, such as the connection string. You then create a `Command` object and provide it with the details of the SQL command that is to be executed. This command can be an inline SQL text command, a stored procedure, or direct table access. You can also provide parameters to these commands if needed. After you create the `Connection` and the `Command` objects, you must decide whether the command returns a result set. If the command doesn't return a result set, you can simply execute the command by calling one of its several `Execute` methods. On the other hand, if the command returns a result set, you must make a decision about whether you want to retain the result set for future uses without maintaining the connection to the database. If you want to retain the result set, you must create a `DataAdapter` object and use it to fill a `DataSet` or a `DataTable` object. These objects are capable of maintaining their information in a disconnected mode. However, if you don't want to retain the result set, but rather to simply process the command in a swift fashion, you can use the `Command` object to create a `DataReader` object. The `DataReader` object needs a live connection to the database, and it works as a forward-only, read-only cursor.

Using the Connection Object

The `Connection` object creates a link (or connection) to a specified data source. This object must contain the necessary information to discover the specified data source and to log in to it properly using a defined username and password combination. This information is provided via a single string called a *connection string*. You can also store this connection string in the `web.config` file of your application.

Every type of data provider has a connection object of some kind. The data provider for working with a SQL data store includes a `SqlConnection` class that performs this type of operation. The `SqlConnection` object is a class that is specific to the `SqlClient` provider. As discussed earlier in this chapter, the `SqlClient` provider is built for working with the SQL Server 7.0 and higher databases. The properties for the `SqlConnection` class are shown in the following table.

Property	Description
`ConnectionString`	This property allows you to read or provide the connection string that should be used by the `SqlConnection` object.
`Database`	This read-only property returns the name of the database to use after the connection is opened.
`Datasource`	This read-only property returns the name of the instance of the SQL Server database used by the `SqlConnection`object.
`State`	This read-only property returns the current state of the connection. The possible values are `Broken`, `Closed`, `Connecting`, `Executing`, `Fetching`, and `Open`.

Connecting to a data source is probably the most common task when you are working with data. This example and the ones that follow assume that you have a SQL Server database. In order to connect to your SQL Server database, you use the `SqlConnection` class. This is shown in Listing 8-5.

Listing 8-5: Connecting to a SQL database

VB

```
Dim conn as SqlConnection
conn = New SqlConnection("Data Source=.\SQLEXPRESS;AttachDbFilename=
           |DataDirectory|\NORTHWND.MDF;Integrated Security=True;
           User Instance=True") ' Put this string on one line in your code
conn.Open()
```

C#

```
SqlConnection conn;
conn = new
         SqlConnection(@"Data Source=.\SQLEXPRESS;AttachDbFilename=
            |DataDirectory|\NORTHWND.MDF;Integrated Security=True;
            User Instance=True");  // Put this string on one line in your code

conn.Open();
```

To make this connection work, be sure that the proper namespaces are imported before you start using any of the classes that work with SQL. The first step in making a connection is to create an instance of the `SqlConnection` class and assign it to the `conn` instance. This `SqlConnection` class is initialized after you pass in the connection string as a parameter to the class. In this case, you are connecting to the Northwind database that resides on your local machine using the system administrator's login credentials.

Another means of making a connection is to put the connection string within the application's `web.config` file and then to make a reference to the `web.config` file. With ASP.NET 3.5, you will find that there is an easy way to manage the storage of your connection strings through the use of the `web.config` file. This is actually a better way to store your connection strings rather than hard-coding them within the code of the application itself. In addition to having a single point in the application where the credentials for database access can be managed, storing credentials in the `web.config` also gives you the ability to encrypt the credentials.

To define your connection string within the `web.config` file, you are going to make use of the `<connectionStrings>` section. From this section, you can place an `<add>` element within it to define your connection. An example of this is illustrated in Listing 8-6.

Listing 8-6: Providing your connection string within the web.config file

```
<connectionStrings>
  <add name="DSN_Northwind" connectionString="Data
   Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\NORTHWND.MDF;Integrated
   Security=True;User Instance=True"
   providerName="System.Data.SqlClient" />
 </connectionStrings>
```

In many places of this chapter, you will see that the actual connection string is broken up on multiple lines. This connection string will need to be on a single line within your code or broken up with string concatenation.

Now that you have a connection string within the `web.config` file, you can then make use of that connection string directly in your code by using the `ConnectionManager` object as illustrated here in Listing 8-7.

Listing 8-7: Using the connection string found in the web.config file

VB

```
conn = New _
   SqlConnection( _
   ConfigurationManager.ConnectionStrings("DSN_Northwind").ConnectionString)
```

C#

```
conn = new
   SqlConnection(
   ConfigurationManager.ConnectionStrings["DSN_Northwind"].ConnectionString);
```

For this line of code to work, you are going to have to make a reference to the `System.Configuration` namespace.

When you complete your connection to the data source, be sure that you close the connection by using `conn.Close()`. The .NET Framework does not implicitly release the connections when they fall out of scope.

Using the Command Object

The `Command` object uses the `Connection` object to execute SQL queries. These queries can be in the form of inline text, stored procedures, or direct table access. If the SQL query uses a `SELECT` clause, the result set it returns is usually stored in either a `DataSet` or a `DataReader` object. The `Command` object provides a number of *Execute* methods that can be used to perform various types of SQL queries.

Next, take a look at some of the more useful properties of the `SqlCommand` class, as shown in the following table.

Property	**Description**
`CommandText`	This read/write property allows you to set or retrieve either the T-SQL statement or the name of the stored procedure.
`CommandTimeout`	This read/write property gets or sets the number of seconds to wait while attempting to execute a particular command. The command is aborted after it times out and an exception is thrown. The default time allotted for this operation is 30 seconds.
`CommandType`	This read/write property indicates the way the `CommandText`property should be interpreted. The possible values are `StoredProcedure`, `TableDirect`, and `Text`. The value of `Text` means that your SQL statement is *inline* or contained within the code itself.
`Connection`	This read/write property gets or sets the `SqlConnection` object that should be used by this `Command` object.

Next, take a look at the various Execute methods that can be called from a `Command` object.

Property	**Description**
`ExecuteNonQuery`	This method executes the command specified and returns the number of rows affected.
`ExecuteReader`	This method executes the command specified and returns an instance of the `SqlDataReader` class. The DataReader object is a read-only and forward-only cursor.
`ExecuteRow`	This method executes the command and returns an instance of the `SqlRecord` class. This object contains only a single returned row.

Property	Description
`ExecuteScalar`	This method executes the command specified and returns the first column of the first row in the form of a generic object. The remaining rows and columns are ignored.
`ExecuteXmlReader`	This method executes the command specified and returns an instance of the `XmlReader` class. This method enables you to use a command that returns the results set in the form of an XML document.

Using the DataReader Object

The `DataReader` object is a simple forward-only and read-only cursor. It requires a live connection with the data source and provides a very efficient way of looping and consuming all or part of the result set. This object cannot be directly instantiated. Instead, you must call the `ExecuteReader` method of the `Command` object to obtain a valid `DataReader` object.

When using a `DataReader` object, be sure to close the connection when you are done using the data reader. If not, then the connection stays alive. The connection utilized stays alive until it is explicitly closed using the `Close()` method or until you have enabled your `Command` object to close the connection. You can close the connection after using the data reader in one of two ways. One way is to provide the `CommandBehavior.CloseConnection` enumeration while calling the `ExecuteMethod` of the `Command` object. This approach works only if you loop through the data reader until you reach the end of the result set, at which point the reader object automatically closes the connection for you. However, if you don't want to keep reading the data reader until the end of the result set, you can call the `Close()` method of the `Connection` object yourself.

Listing 8-8 shows the `Connection`, `Command`, and `DataReader` objects in action. It shows how to connect to the Northwind database (an example database found in the Microsoft's SQL Server 7.0, 2000, 2005, or 2008 database servers), read the Customers table within this database, and display the results in a GridView server control.

Listing 8-8: The SqlConnection, SqlCommand, and SqlDataReader objects in action

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, _
            ByVal e As System.EventArgs)

        If Not Page.IsPostBack Then
            Dim MyReader As SqlDataReader

     Dim MyConnection As SqlConnection = New SqlConnection()
```

Continued

```
            MyConnection.ConnectionString = _
        ConfigurationManager.ConnectionStrings("DSN_Northwind").ConnectionString

            Dim MyCommand As SqlCommand = New SqlCommand()
            MyCommand.CommandText = "SELECT TOP 3 * FROM CUSTOMERS"
            MyCommand.CommandType = CommandType.Text
            MyCommand.Connection = MyConnection

            MyCommand.Connection.Open()
            MyReader = MyCommand.ExecuteReader(CommandBehavior.CloseConnection)

            gvCustomers.DataSource = MyReader
            gvCustomers.DataBind()

            MyCommand.Dispose()
            MyConnection.Dispose()
        End If
    End Sub
</script>

<html>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:GridView ID="gvCustomers" runat="server">
        </asp:GridView>
    </div>
    </form>
</body>
</html>
```

C#

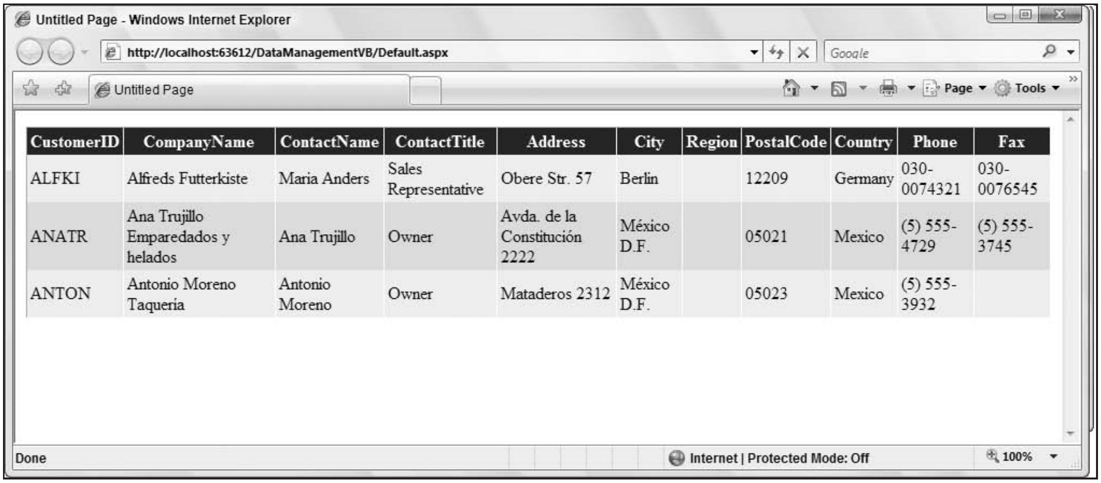
```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            SqlDataReader MyReader;
            SqlConnection MyConnection = new SqlConnection();
            MyConnection.ConnectionString =
        ConfigurationManager.ConnectionStrings["DSN_Northwind"].ConnectionString;

            SqlCommand MyCommand = new SqlCommand();
            MyCommand.CommandText = "SELECT TOP 3 * FROM CUSTOMERS";
            MyCommand.CommandType = CommandType.Text;
            MyCommand.Connection = MyConnection;

            MyCommand.Connection.Open();
```

Continued

```
            MyReader = MyCommand.ExecuteReader(CommandBehavior.CloseConnection);

            gvCustomers.DataSource = MyReader;
            gvCustomers.DataBind();

            MyCommand.Dispose();
            MyConnection.Dispose();
        }
    }
</script>
```

The code shown in Listing 8-8 uses the `SqlConnection` class to create a connection with the Northwind database using the connection string stored in the `web.config` file. This connection string is then retrieved using the `ConfigurationManager` class. It is always best to store your connection strings inside the `web.config` and to reference them in this manner. If you have a single place to work with your connection strings, any task is a lot more manageable than if you place all your connection strings in the actual code of your application.

After working with the connection string, this bit of code from Listing 8-8 creates a `Command` object using the `SqlCommand` class because you are interested in working with a SQL database. Next, the code provides the command text, command type, and connection properties. After the command and the connection are created, the code opens the connection and executes the command by calling the `ExecuteReader` method of the `MyCommand` object. After receiving the data reader from the `Command` object, you simply bind the retrieved results to an instance of the GridView control. The results are shown in Figure 8-1.


Untitled Page - Windows Internet Explorer
http://localhost:63612/DataManagementVB/Default.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076545
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	

Figure 8-1

Using Data Adapter

The `SqlDataAdapter` is a special class whose purpose is to bridge the gap between the disconnected `DataTable` objects and the physical data source. The `SqlDataAdapter` provides a two-way data transfer mechanism. It is capable of executing a `SELECT` statement on a data source and transferring the result set into a `DataTable` object. It is also capable of executing the standard `INSERT`, `UPDATE`, and `DELETE` statements and extracting the input data from a `DataTable` object.

The commonly used properties offered by the `SqlDataAdapter` class are shown in the following table.

Property	Description
`SelectCommand`	This read/write property sets or gets an object of type `SqlCommand`. This command is automatically executed to fill a `DataTable` with the result set.
`InsertCommand`	This read/write property sets or gets an object of type `SqlCommand`. This command is automatically executed to insert a new record to the SQL Server database.
`UpdateCommand`	This read/write property sets or gets an object of type `SqlCommand`. This command is automatically executed to update an existing record on the SQL Server database.
`DeleteCommand`	This read/write property sets or gets an object of type `SqlCommand`. This command is automatically executed to delete an existing record on the SQL Server database.

The `SqlDataAdapter` class also provides a method called `Fill()`. Calling the `Fill()` method automatically executes the command provided by the `SelectCommand` property, receives the result set, and copies it to a `DataTable` object.

The code example in Listing 8-9 illustrates how to use an object of `SqlDataAdapter` class to fill a `DataTable` object.

Listing 8-9: Using an object of SqlDataAdapter to fill a DataTable

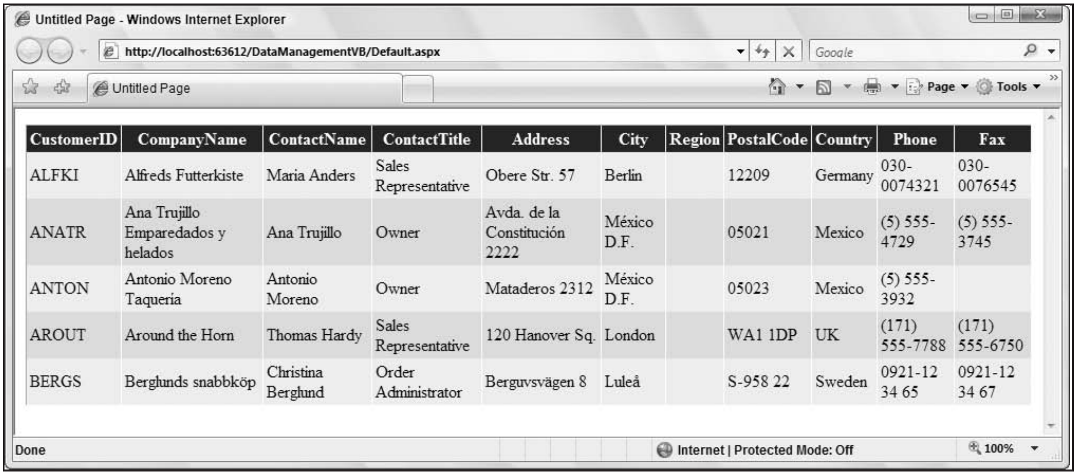
VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, _
            ByVal e As System.EventArgs)

        If Not Page.IsPostBack Then
            Dim MyTable As DataTable = New DataTable()

            Dim MyConnection As SqlConnection = New SqlConnection()
            MyConnection.ConnectionString = _
            ConfigurationManager.ConnectionStrings("DSN_Northwind").ConnectionString

            Dim MyCommand As SqlCommand = New SqlCommand()
            MyCommand.CommandText = "SELECT TOP 5 * FROM CUSTOMERS"
            MyCommand.CommandType = CommandType.Text
            MyCommand.Connection = MyConnection

            Dim MyAdapter As SqlDataAdapter = New SqlDataAdapter()
            MyAdapter.SelectCommand = MyCommand
            MyAdapter.Fill(MyTable)

            gvCustomers.DataSource = MyTable.DefaultView
            gvCustomers.DataBind()

            MyAdapter.Dispose()
            MyCommand.Dispose()
            MyConnection.Dispose()
        End If

    End Sub
</script>
```

C#

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            DataTable MyTable = new DataTable();

            SqlConnection MyConnection = new SqlConnection();
            MyConnection.ConnectionString =
                ConfigurationManager.
                ConnectionStrings["DSN_Northwind"].ConnectionString;

            SqlCommand MyCommand = new SqlCommand();
            MyCommand.CommandText = "SELECT TOP 5 * FROM CUSTOMERS";
            MyCommand.CommandType = CommandType.Text;
            MyCommand.Connection = MyConnection;

            SqlDataAdapter MyAdapter = new SqlDataAdapter();
            MyAdapter.SelectCommand = MyCommand;
            MyAdapter.Fill(MyTable);

            gvCustomers.DataSource = MyTable.DefaultView;
            gvCustomers.DataBind();

            MyAdapter.Dispose();
            MyCommand.Dispose();
            MyConnection.Dispose();        }
    }
</script>
```

The code shown in Listing 8-9 creates a `Connection` and `Command` object and then proceeds to create an instance of the `SqlDataAdapter` class. It then sets the `SelectCommand` property of the `DataAdapter` object to the `Command` object it had previously created. After the `DataAdapter` object is ready for executing, the code executes the `Fill()` method, passing it an instance of the `DataTable` class. The `Fill()` method populates the `DataTable` object. Figure 8-2 shows the result of executing this code.


Untitled Page - Windows Internet Explorer
http://localhost:63612/DataManagementVB/Default.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076545
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK	(171) 555-7788	(171) 555-6750
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65	0921-12 34 67

Figure 8-2

Using Parameters

Most serious database programming, regardless of how simple it might be, requires you to configure SQL statements using parameters. Using parameters helps guard against possible SQL injection attacks. Obviously, a discussion on the basics of ADO.NET programming is not complete without covering the use of parameterized SQL statements.

Creating a parameter is as simple as declaring an instance of the `SqlParameter` class and providing it the necessary information, such as parameter name, value, type, size, direction, and so on. The following table shows the properties of the `SqlParameter` class.

Property	Description
`ParameterName`	This read/write property gets or sets the name of the parameter.
`SqlDbType`	This read/write property gets or sets the SQL Server database type of the parameter value.
`Size`	This read/write property sets or gets the size of the parameter value.
`Direction`	This read/write property sets or gets the direction of the parameter, such as `Input`, `Output`, or `InputOutput`.
`SourceColumn`	This read/write property maps a column from a `DataTable` to the parameter. It enables you to execute multiple commands using the `SqlDataAdapter` object and pick the correct parameter value from a `DataTable` column during the command execution.
`Value`	This read/write property sets or gets the value provided to the parameter object. This value is passed to the parameter defined in the command during runtime.

Listing 8-10 modifies the code shown in Listing 8-5 to use two parameters while retrieving the list of customers from the database.

Listing 8-10: The use of a parameterized SQL statement

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, _
            ByVal e As System.EventArgs)
        If Not Page.IsPostBack Then
            Dim MyReader As SqlDataReader
            Dim CityParam As SqlParameter
            Dim ContactParam As SqlParameter

            Dim MyConnection As SqlConnection = New SqlConnection()
            MyConnection.ConnectionString = _
         ConfigurationManager.ConnectionStrings("DSN_Northwind").ConnectionString

            Dim MyCommand As SqlCommand = New SqlCommand()
            MyCommand.CommandText = _
         "SELECT * FROM CUSTOMERS WHERE CITY = @CITY AND CONTACTNAME = @CONTACT"
            MyCommand.CommandType = CommandType.Text
            MyCommand.Connection = MyConnection

            CityParam = New SqlParameter()
            CityParam.ParameterName = "@CITY"
            CityParam.SqlDbType = SqlDbType.VarChar
            CityParam.Size = 15
            CityParam.Direction = ParameterDirection.Input
            CityParam.Value = "Berlin"

            ContactParam = New SqlParameter()
            ContactParam.ParameterName = "@CONTACT"
            ContactParam.SqlDbType = SqlDbType.VarChar
            ContactParam.Size = 15
            ContactParam.Direction = ParameterDirection.Input
            ContactParam.Value = "Maria Anders"

            MyCommand.Parameters.Add(CityParam)
            MyCommand.Parameters.Add(ContactParam)

            MyCommand.Connection.Open()
            MyReader = MyCommand.ExecuteReader(CommandBehavior.CloseConnection)

            gvCustomers.DataSource = MyReader
            gvCustomers.DataBind()

            MyCommand.Dispose()
            MyConnection.Dispose()
        End If
```

Continued

```
    End Sub
</script>
```

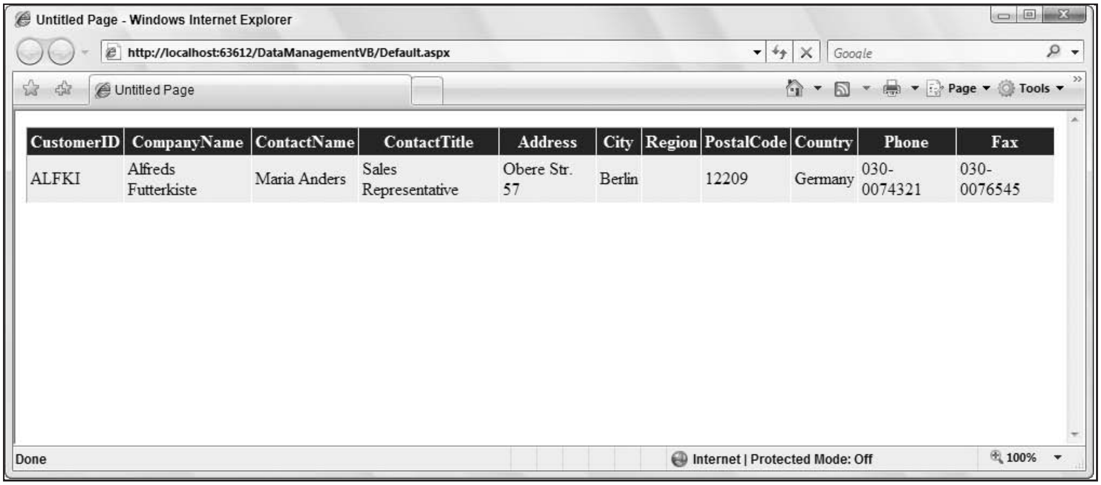
C#

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
       if (!Page.IsPostBack)
       {
            SqlDataReader MyReader;
            SqlParameter CityParam;
            SqlParameter ContactParam;

            SqlConnection MyConnection = new SqlConnection();
            MyConnection.ConnectionString =
        ConfigurationManager.ConnectionStrings["DSN_Northwind"].ConnectionString;

            SqlCommand MyCommand = new SqlCommand();
            MyCommand.CommandText =
        "SELECT * FROM CUSTOMERS WHERE CITY = @CITY AND CONTACTNAME = @CONTACT";
            MyCommand.CommandType = CommandType.Text;
            MyCommand.Connection = MyConnection;

            CityParam = new SqlParameter();
            CityParam.ParameterName = "@CITY";
            CityParam.SqlDbType = SqlDbType.VarChar;
            CityParam.Size = 15;
            CityParam.Direction = ParameterDirection.Input;
            CityParam.Value = "Berlin";

            ContactParam = new SqlParameter();
            ContactParam.ParameterName = "@CONTACT";
            ContactParam.SqlDbType = SqlDbType.VarChar;
            ContactParam.Size = 15;
            ContactParam.Direction = ParameterDirection.Input;
            ContactParam.Value = "Maria Anders";

            MyCommand.Parameters.Add(CityParam);
            MyCommand.Parameters.Add(ContactParam);

            MyCommand.Connection.Open();
            MyReader = MyCommand.ExecuteReader(CommandBehavior.CloseConnection);

            gvCustomers.DataSource = MyReader;
            gvCustomers.DataBind();
```

Continued

```
                MyCommand.Dispose();
                MyConnection.Dispose();
            }
        }
    </script>
```

The code shown in Listing 8-8 uses a parameterized SQL statement that receives the name of the city and the contact person to narrow the result set. These parameters are provided by instantiating a couple of instances of the `SqlParameter` class and filling in the appropriate name, type, size, direction, and value properties for each object of `SqlParameter` class. From there, you add the populated parameters to the `Command` object by invoking the `Add()` method of the `Parameters` collection. The result of executing this code is shown in Figure 8-3.


Untitled Page - Windows Internet Explorer
http://localhost:63612/DataManagementVB/Default.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076545

Figure 8-3

Understanding DataSet and DataTable

Most programmers agree that the `DataSet` class is the most commonly used part of ADO.NET in real-world, database-driven applications. This class provides mechanisms for managing data when it is disconnected from the data source. This capability to handle data in a disconnected state was first introduced in .NET during the 1.0 version of ADO.NET. The current 3.5 version of ADO.NET retains all the features of its predecessors and provides a few newer, much needed features.

An object created from the `DataSet` class works as a container for other objects that are created from the `DataTable` class. The `DataTable` object represents a logical table in memory. It contains rows, columns, primary keys, constraints, and relations with other `DataTable` objects. Therefore, you could have a `DataSet` that is made up of two distinct tables such as a Customers and an Orders table. Then you could use the `DataSet`, just as you would any other relational data source, to make a relation between the two tables in order to show all the orders for a particular customer.

Most of the disconnected data-driven programming is actually done using one or more `DataTable` objects within the `DataSet`. However, the previous versions of ADO.NET didn't allow you to work directly with the `DataTable` object for some very important tasks, such as reading and writing data to and from an

XML file. It didn't even allow you to serialize the `DataTable` object independently of the larger and encompassing `DataSet` object. This limitation required you to always use the `DataSet` object to perform any operation on a `DataTable`. The current version of ADO.NET removes this limitation and enables you to work directly with the `DataTable` for all your needs. In fact, we recommend that you don't use the `DataSet` object unless you need to work with multiple `DataTable` objects and need a container object to manage them. If you end up working with only a single table of information, then it is best to work with an instance of the `DataTable` object rather than a `DataSet` that contains only a single `DataTable`.

The current version of ADO.NET provides the capability to load a `DataTable` in memory by consuming a data source using a `DataReader`. In the past, you were sometimes restricted to creating multiple overloads of the same method just to work with both the `DataReader` and the `DataTable` objects. Now you have the flexibility to write the data access code one time and reuse the `DataReader` — either directly or to fill a `DataTable`, as shown in Listing 8-11.

Listing 8-11: How to load a DataTable from a DataReader

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, _
            ByVal e As System.EventArgs)

        If Not Page.IsPostBack Then
            Dim MyDataTable As DataTable
            Dim MyReader As SqlDataReader
            Dim CityParam As SqlParameter

            Dim MyConnection As SqlConnection = New SqlConnection()
            MyConnection.ConnectionString = _
        ConfigurationManager.ConnectionStrings("DSN_Northwind").ConnectionString

            Dim MyCommand As SqlCommand = New SqlCommand()
            MyCommand.CommandText = _
                    "SELECT * FROM CUSTOMERS WHERE CITY = @CITY"
            MyCommand.CommandType = CommandType.Text
            MyCommand.Connection = MyConnection

            CityParam = New SqlParameter()
            CityParam.ParameterName = "@CITY"
            CityParam.SqlDbType = SqlDbType.VarChar
            CityParam.Size = 15
            CityParam.Direction = ParameterDirection.Input
            CityParam.Value = "London"

            MyCommand.Parameters.Add(CityParam)

            MyCommand.Connection.Open()
```

Continued

```
            MyReader = MyCommand.ExecuteReader(CommandBehavior.CloseConnection)

            MyDataTable = New DataTable()

            ' Loading DataTable using a DataReader
            MyDataTable.Load(MyReader)

            gvCustomers.DataSource = MyDataTable
            gvCustomers.DataBind()

            MyDataTable.Dispose()
            MyCommand.Dispose()
            MyConnection.Dispose()
        End If

    End Sub

</script>
```

C#

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack )
        {
            DataTable MyDataTable;
            SqlDataReader MyReader;
            SqlParameter CityParam;

            SqlConnection MyConnection = new SqlConnection();
            MyConnection.ConnectionString =
        ConfigurationManager.ConnectionStrings["DSN_Northwind"].ConnectionString;

            SqlCommand MyCommand = new SqlCommand();
            MyCommand.CommandText =
               "SELECT * FROM CUSTOMERS WHERE CITY = @CITY";
            MyCommand.CommandType = CommandType.Text;
            MyCommand.Connection = MyConnection;

            CityParam = new SqlParameter();
            CityParam.ParameterName = "@CITY";
            CityParam.SqlDbType = SqlDbType.VarChar;
            CityParam.Size = 15;
            CityParam.Direction = ParameterDirection.Input;
            CityParam.Value = "London";

            MyCommand.Parameters.Add(CityParam);
```

Continued

```
            MyCommand.Connection.Open();
            MyReader = MyCommand.ExecuteReader(CommandBehavior.CloseConnection);

            MyDataTable = new DataTable();

            // Loading DataTable using a DataReader
            MyDataTable.Load(MyReader);

            gvCustomers.DataSource = MyDataTable;
            gvCustomers.DataBind();

            MyDataTable.Dispose();
            MyCommand.Dispose();
            MyConnection.Dispose();
        }
    }
</script>
```

Not only can you load a `DataTable` object from a `DataReader` object, you can also retrieve a `DataTableReader` from an existing `DataTable` object. This is accomplished by calling the `CreateDataReader` method of the `DataTable` class. This method returns an instance of the `DataTableReader` object that can be passed to any method that expects to receive a `DataReader`.

Deciding When to Use a DataSet

As revolutionary as a `DataSet` might be, it is not the best choice in every situation. Often, it may not be appropriate to use a `DataSet`; instead it might be better to use a `DataReader`.

With ADO 2.6, it was possible to perform a command upon a data store and get back a single collection of data made up of any number of rows. You could then iterate through this collection of data and use it in some fashion. Now ADO.NET can use the `DataSet` to return a collection of data that actually keeps its structure when removed from the data store. In some situations, you benefit greatly from keeping this copy in its original format. By doing so, you can keep the data disconnected in an in-memory cache in its separate tables and work with the tables individually or apply relationships between the tables. You can work with the tables in much the same manner as you do with other relational data sources — using a parent/child relationship. If it is to your advantage to work with certain data with all its relationships in place (in order to enforce a parent/child relationship upon the data); in this case, of course, it is better to use a `DataSet` as opposed to a `DataReader`.

Because the `DataSet` is a disconnected copy of the data, you can work with the same records repeatedly without having to go back to the data store. This capability can greatly increase performance and lessen the load upon the server. Having a copy of the data separate from the data store also enables you to continuously handle and shape the data locally. For instance, you might need to repeatedly filter or sort through a collection of data. In this case, it would be of great advantage to work with a `DataSet` rather than going back and forth to the data store itself.

Probably one of the greatest uses of the `DataSet` is to work with multiple data stores and come away with a single collection of data. So for instance, if you have your Customers table within SQL and the orders information for those particular customers within an Oracle database, you can very easily query each data store and create a single `DataSet` with a Customers and an Orders table in place that you can use in any fashion you choose. The DataSet is just a means of storage for data and doesn't concern itself with where the data came from. So, if you are working with data that is coming from multiple data stores, it is to your benefit to use the `DataSet`.

Because the `DataSet` is based upon XML and XML Schemas, it is quite easy to move the `DataSet` around — whether you are transporting it across tiers, processes or between disparate systems or applications. If the application or system to which you are transferring the `DataSet` doesn't understand `DataSet`s, the `DataSet` represents itself as an XML file. So basically, any system or application that can interpret and understand XML can work with the `DataSet`. This makes it a very popular transport vehicle, and you see an example of it when you transport the `DataSet` from an XML Web service.

Last but not least, the `DataSet` enables you to program data with ease. It is much simpler than anything that has been provided before the .NET Framework came to the scene. Putting the data within a class object allows you to programmatically access the `DataSet`. The code example in Listing 8-12 shows you just how easy it can be.

Listing 8-12: An example of working with the DataSet object

VB

```
Dim conn As SqlConnection = New SqlConnection _
   (ConfigurationManager.ConnectionStrings("DSN_Northwind").ConnectionString)
conn.Open()
Dim da As SqlDataAdapter = New SqlDataAdapter("Select * from Customers", conn)
Dim ds As DataSet = New DataSet()
da.Fill(ds, "CustomersTable")
```

C#

```
SqlConnection conn = new SqlConnection
   (ConfigurationManager.ConnectionStrings["DSN_Northwind"].ConnectionString);
conn.Open();
SqlDataAdapter da = new SqlDataAdapter("Select * from Customers", conn);
DataSet ds = new DataSet();
da.Fill(ds, "CustomersTable");
```

Basically, when you work with data, you have to weigh when to use the `DataSet`. In some cases, you get extreme benefits from using this piece of technology that is provided with ADO.NET. Sometimes, however, you may find it is not in your best interests to use the `DataSet`. Instead, it is best to use the `DataReader`.

The `DataSet` can be used whenever you choose, but sometimes you would rather use the `DataReader` and work directly against the data store. By using the command objects, such as the `SqlCommand` and the `OleDbCommand` objects, you have a little more direct control over what is executed and what you get back as a result set. In situations where this is vital, it is to your advantage not to use the `DataSet`.

When you don't use the `DataSet`, you don't incur the cost of extra overhead because you are reading and writing directly to the data source. Performing operations in this manner means you don't have to instantiate any additional objects — avoiding unnecessary steps.

This is especially true in a situation when you work with Web Forms in ASP.NET. If you are dealing with Web Forms, the Web pages are re-created each and every time. When this happens, not only is the page re-created by the call to the data source, the `DataSet` is also re-created unless you are caching the `DataSet` in some fashion. This can be an expensive process; so, in situations such as this, you might find it to your benefit to work directly off the data source using the `DataReader`. In most situations when you are working with Web Forms, you want to work with the `DataReader` instead of creating a `DataSet`.

The Typed DataSet

As powerful as the `DataSet` is, it still has some limitations. The `DataSet` is created at runtime. It accesses particular pieces of data by making certain assumptions. Take a look at how you normally access a specific field in a `DataSet` that is not strongly typed (Listing 8-13).

Listing 8-13: Accessing a field in a DataSet

VB

```
ds.Tables("Customers").Rows(0).Columns("CompanyName") = "XYZ Company"
```

C#

```
ds.Tables["Customers"].Rows[0].Columns["CompanyName"] = "XYZ Company";
```

The preceding code looks at the Customers table, the first row (remember, everything is zero-based) in the column `CompanyName`, and assigns the value of `XYZ Company` to the field. This is pretty simple and straightforward, but it is based upon certain assumptions and is generated at runtime. The `"Customers"` and `"CompanyName"` words are string literals in this line of code. If they are spelled wrong or if these items aren't in the table, an error occurs at runtime.

Listing 8-14 shows you how to assign the same value to the same field by using a typed `DataSet`.

Listing 8-14: Accessing a field in a typed DataSet

VB

```
ds.Customers(0).CompanyName = "XYZ Company"
```

C#

```
ds.Customers[0].CompanyName = "XYZ Company";
```

Now the table name and the field to be accessed are not treated as string literals but, instead, are encased in an XML Schema and a class that is generated from the `DataSet` class. When you create a typed `DataSet`, you are creating a class that implements the tables and fields based upon the schema used to generate the class. Basically, the schema is coded into the class.

As you compare the two examples, you see that a typed `DataSet` is easier to read and understand. It is less error-prone, and errors are realized at compile time as opposed to runtime.

In the end, typed `DataSet`s are optional, and you are free to use either style as you code.

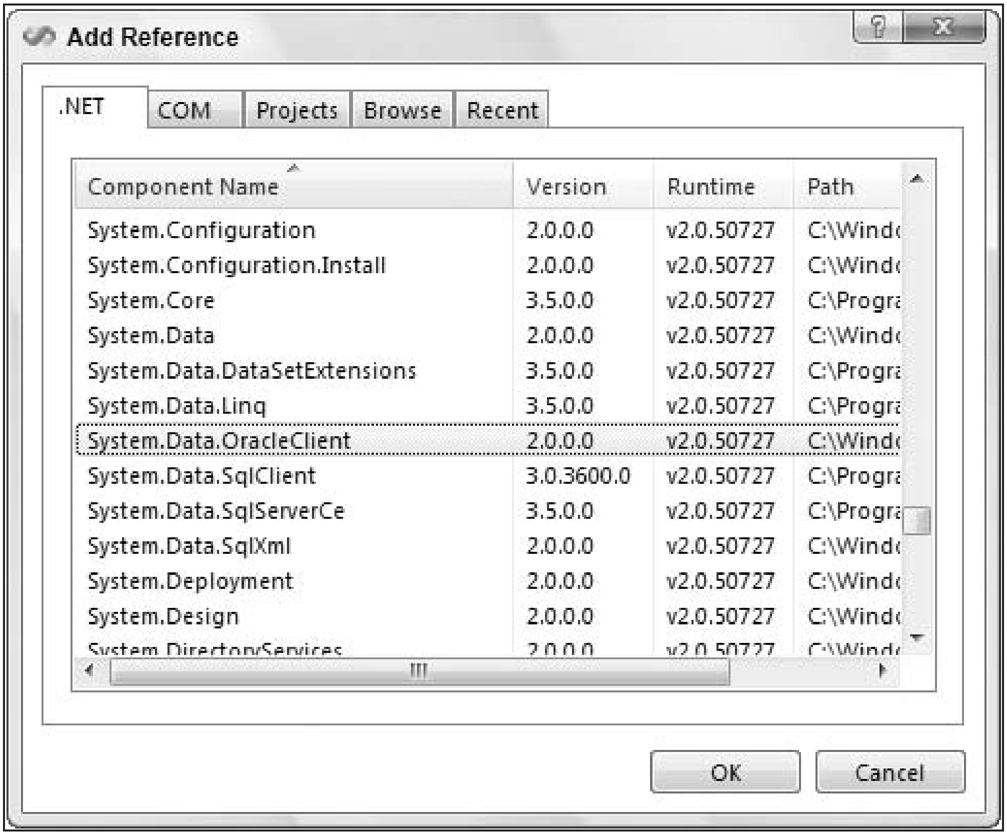
Using Oracle as Your Database with ASP.NET 3.5

If you work in the enterprise space, in many cases you must work with an Oracle back-end database. ADO.NET 2.0 has a built-in capability to work with Oracle using the `System.Data.OracleClient` namespace.

First, in order to connect ASP.NET to your Oracle database, you install the Oracle 10 g Client on your Web server. You can get this piece of software from the Oracle Web site found at `oracle.com`. If you are able to connect to your Oracle database from your Web server using SQL*Plus (an Oracle IDE for working with an Oracle database), can use the Microsoft-built Oracle data provider, `System.Data.OracleClient`.

> *If you are still having trouble connecting to your Oracle database, you also may try to make sure that the database connection is properly defined in your server's .ora file found at* `C:\Oracle\product\10.1.0\Client_1\NETWORK\ADMIN`*. Note that the version number might be different.*

After you know you can connect to Oracle, you can make use of the Microsoft-built Oracle data provider. To utilize the built-in capabilities to connect to Oracle, your ASP.NET application must reference this DLL. To do this, right-click your project in the Visual Studio Solution Explorer and select Add Reference from the list of options presented. This gives you a long list of available .NET components. Select the `System.Data.OracleClient component`. Notice the two versions of this component (as illustrated in Figure 8-4). You select the one that is built for the .NET Framework 2.0.



Component Name	Version	Runtime	Path
System.Configuration	2.0.0.0	v2.0.50727	C:\Wind
System.Configuration.Install	2.0.0.0	v2.0.50727	C:\Wind
System.Core	3.5.0.0	v2.0.50727	C:\Progra
System.Data	2.0.0.0	v2.0.50727	C:\Wind
System.Data.DataSetExtensions	3.5.0.0	v2.0.50727	C:\Progra
System.Data.Linq	3.5.0.0	v2.0.50727	C:\Progra
System.Data.OracleClient	2.0.0.0	v2.0.50727	C:\Wind
System.Data.SqlClient	3.0.3600.0	v2.0.50727	C:\Progra
System.Data.SqlServerCe	3.5.0.0	v2.0.50727	C:\Progra
System.Data.SqlXml	2.0.0.0	v2.0.50727	C:\Wind
System.Deployment	2.0.0.0	v2.0.50727	C:\Wind
System.Design	2.0.0.0	v2.0.50727	C:\Wind
System.DirectoryServices	2.0.0.0	v2.0.50727	C:\Wind

Figure 8-4

After this is added, you find the reference to this component in the `web.config` file of your ASP.NET application (as presented in Listing 8-15).

Listing 8-15: The reference to the System.Data.OracleClient DLL in the web.config

```
<configuration>
   <system.web>
      <compilation debug="true">
         <assemblies>
            <add assembly="System.Data.OracleClient,
                Version=2.0.0.0, Culture=neutral,
                PublicKeyToken=B77A5C561934E089"/>
         </assemblies>
      </compilation>
   </system.web>
</configuration>
```

With this reference in place, you also reference this available DLL in your page along with `System.Data`. This is demonstrated in Listing 8-16.

Listing 8-16: Referencing the System.Data.OracleClient DLL

VB

```
Imports System.Data
Imports System.Data.OracleClient
```

C#

```
using System.Data;
using System.Data.OracleClient;
```

With all the references in place, you are able to work with an Oracle backend in pretty much the same manner as you work with a SQL Server backend. Listing 8-17 shows you just how similar it is.

Listing 8-17: Using the OracleClient object to connect to an Oracle database

VB

```
Dim conn As OracleConnection
Dim cmd As OracleCommand

Dim cmdString As String = "Select CompanyName from Customers"
conn = New _
   OracleConnection("User Id=bevjen;Password=bevjen01;Data Source=myOracleDB")
cmd = New OracleCommand(cmdString, conn)
cmd.CommandType = CommandType.Text

conn.Open()
```

C#

```
OracleConnection conn;
OracleCommand cmd;

string cmdString = "Select CompanyName from Customers";
conn = new
   OracleConnection("User Id=bevjen;Password=bevjen01;Data Source=myOracleDB");
cmd = new OracleCommand(cmdString, conn);
```

```
cmd.CommandType = CommandType.Text;

conn.Open();
```

After you are connected and performing the PL-SQL commands you want, you can use the `OracleDataReader` object just as you would use the `SqlDataReader` object.

> *Notice that, in this section, I have made reference to the Microsoft-built Oracle data provider. Another option, and many developers consider this the better option, is to use the Oracle-built ODP.NET data provider instead. This data provider can be freely downloaded from the Oracle download page at* `oracle.com`. *You can then reference this new DLL in your project. It is now simply a matter of importing and working with* `System.DataAccess.OracleClient` *in your applications. The Oracle-built data provider contains the capability to work with the more advanced feature provided from the Oracle 10g database.*

The DataList Server Control

The DataList control has been around since the beginning of ASP.NET. It is part of a series of controls that enable you to display your data (especially repeated types of data) using templates. Templates enable you to create more sophisticated layouts for your data and perform functionss that controls such as the GridView server control cannot.

Template-based controls like the DataList control require more work on your part. For instance, you have to build common tasks for yourself. You cannot rely on other data controls, which you might be used to, such s as paging.

Looking at the Available Templates

The idea, when using template-based controls such as the DataList control, is that you put together specific templates to create your desired detailed layout. The DataList control has a number of templates that you can use to build your display. The available templates are defined here in the following table:

Template	Description
`AlternatingItemTemplate`	Works in conjunction with the `ItemTemplate` to provide a layout for all the odd rows within the layout. This is commonly used if you want to have a grid or layout where each row is distinguished in some way (such as having a different background color).
`EditItemTemplate`	Allows for a row or item to be defined on how it looks and behaves when editing.
`FooterTemplate`	Allows the last item in the template to be defined. If this is not defined, then no footer will be used.
`HeaderTemplate`	Allows the first item in the template to be defined. If this is not defined, then no header will be used.
`ItemTemplate`	The core template that is used to define a row or layout for each item in the display.

Template	Description
`SelectedItemTemplate`	Allows for a row or item to be defined on how it looks and behaves when selected.
`SeparatorTemplate`	The layout of any separator that is used between the items in the display.

Working with ItemTemplate

Although you have seven templates available to you for use with the DataList control, at a minimum, you are going to need the ItemTemplate. The following example, shown here in Listing 8-18, shows the company names from the Northwind database.

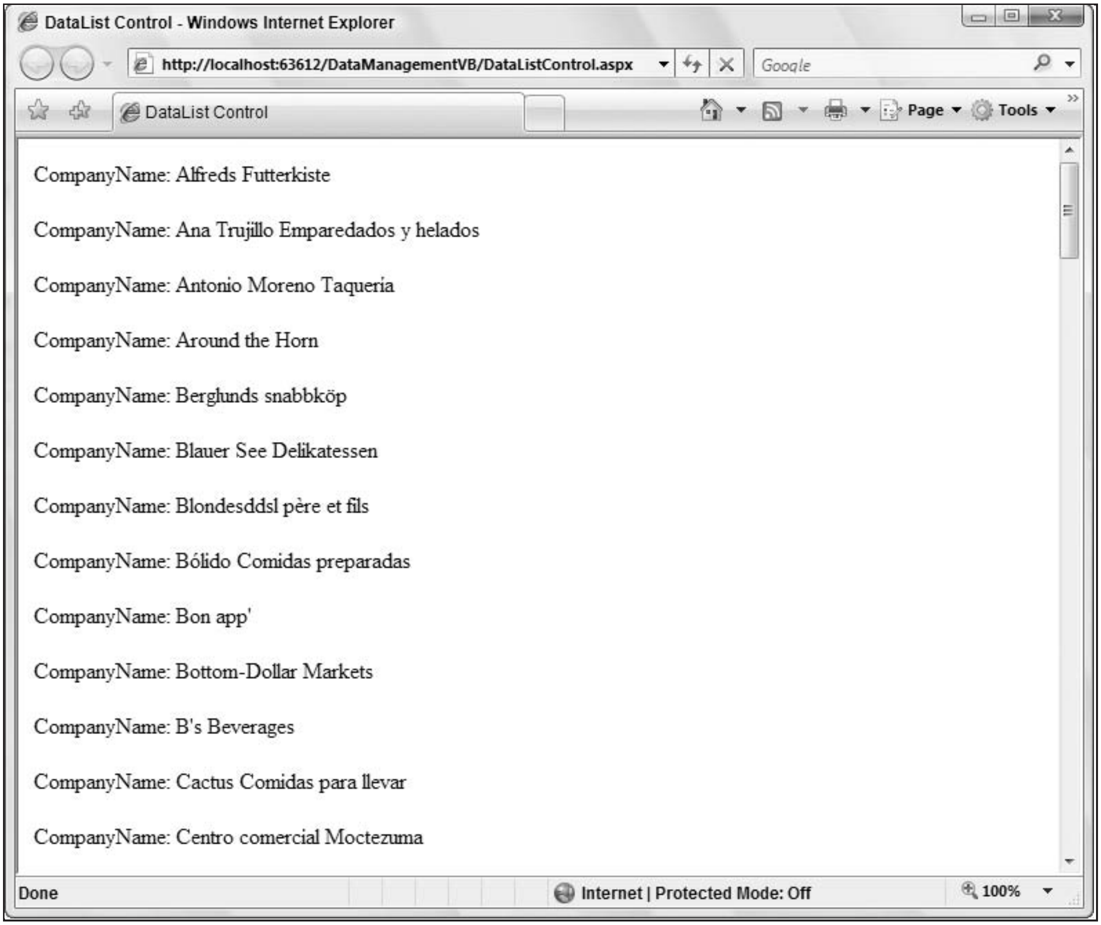
Listing 8-18: Showing the company names from the Northwind database using DataList

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="DataListControl.aspx.vb"
    Inherits="DataListControl" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>DataList Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:DataList ID="DataList1" runat="server" DataSourceID="SqlDataSource1">
            <ItemTemplate>
                Company Name:
                <asp:Label ID="CompanyNameLabel" runat="server"
                    Text='<%# Eval("CompanyName") %>' />
                <br />
                <br />
            </ItemTemplate>
        </asp:DataList>
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:DSN_Northwind %>"
            SelectCommand="SELECT [CompanyName] FROM [Customers]">
        </asp:SqlDataSource>
    </div>
    </form>
</body>
</html>
```

As stated, the DataList control requires, at a minimum, an `ItemTemplate` element where you define the page layout for each item that is encountered from the data source. In this case, all the data is pulled from the Northwind database sample using the SqlDataSource control. The SqlDataSource control pulls only the CompanyName column from the Customers table. From there, the ItemTemplate section of the DataList control defines two items within it. The first item is a static item, ‘‘Company Name:’’ followed by a single ASP.NET server control, the Label server control. Second, the item is then followed by a couple of standard HTML elements. The `Text` property of the Label control uses inline data binding (as shown in the previous chapter of this book) to bind the values that are coming out of the SqlDataSource control. If there were more than one data point coming out of the SqlDataSource control, you can still specifically grab the data point that you are interested in using by specifying the item in the `Eval` statement.

```
<asp:Label ID="CompanyNameLabel" runat="server"
 Text='<%# Eval("CompanyName") %>' />
```

Using the code from Listing 8-18 gives you the following results as illustrated in Figure 8-5.


DataList Control - Windows Internet Explorer
http://localhost:63612/DataManagementVB/DataListControl.aspx
Google
DataList Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

CompanyName: Alfreds Futterkiste

CompanyName: Ana Trujillo Emparedados y helados

CompanyName: Antonio Moreno Taquería

CompanyName: Around the Horn

CompanyName: Berglunds snabbköp

CompanyName: Blauer See Delikatessen

CompanyName: Blondesddsl père et fils

CompanyName: Bólido Comidas preparadas

CompanyName: Bon app'

CompanyName: Bottom-Dollar Markets

CompanyName: B's Beverages

CompanyName: Cactus Comidas para llevar

CompanyName: Centro comercial Moctezuma

Figure 8-5

If you then look at the source of the page, you can see that the DataList control uses tables by default to lay out the elements.

```
<table id="DataList1" cellspacing="0" border="0" style="border-collapse:collapse;">
   <tr>
      <td>
         CompanyName:
            <span id="DataList1_ctl00_CompanyNameLabel">Alfreds Futterkiste</span>
            <br />
            <br />
         </td>
      </tr><tr>
         <td>
            CompanyName:
            <span id="DataList1_ctl01_CompanyNameLabel">
             Ana Trujillo Emparedados y helados</span>
            <br />
            <br />
         </td>
      </tr>

      <!-- Code removed for clarity -->

   </table>
```

Although this table layout is the default, you can change this so that the DataList control outputs `<span>` tags instead. This is done through the use of the `RepeatLayout` property of the DataList control. You will need to rework your DataList, as is shown in Listing 8-19.

Listing 8-19: Changing the output style using RepeatLayout

```
<asp:DataList ID="DataList1" runat="server" DataSourceID="SqlDataSource1"
 RepeatLayout="Flow">
   <ItemTemplate>
    Company Name:
    <asp:Label ID="CompanyNameLabel" runat="server"
     Text='<%# Eval("CompanyName") %>' />
    <br />
    <br />
   </ItemTemplate>
</asp:DataList>
```

The possible options for the `RepeatLayout` property are either `Table` or `Flow`. `Table` is the default setting. The output you will get when looking at the source of the page when using the `Flow` setting is presented here:

```
<span id="DataList1">
   <span>
      CompanyName:
      <span id="DataList1_ctl00_CompanyNameLabel">Alfreds Futterkiste</span>
         <br />
         <br />
      </span><br />
   <span>
       CompanyName:
       <span id="DataList1_ctl01_CompanyNameLabel">
        Ana Trujillo Emparedados y helados</span>
       <br />
       <br />
   </span>
```

```
        <!-- Code removed for clarity -->

    </span>
```

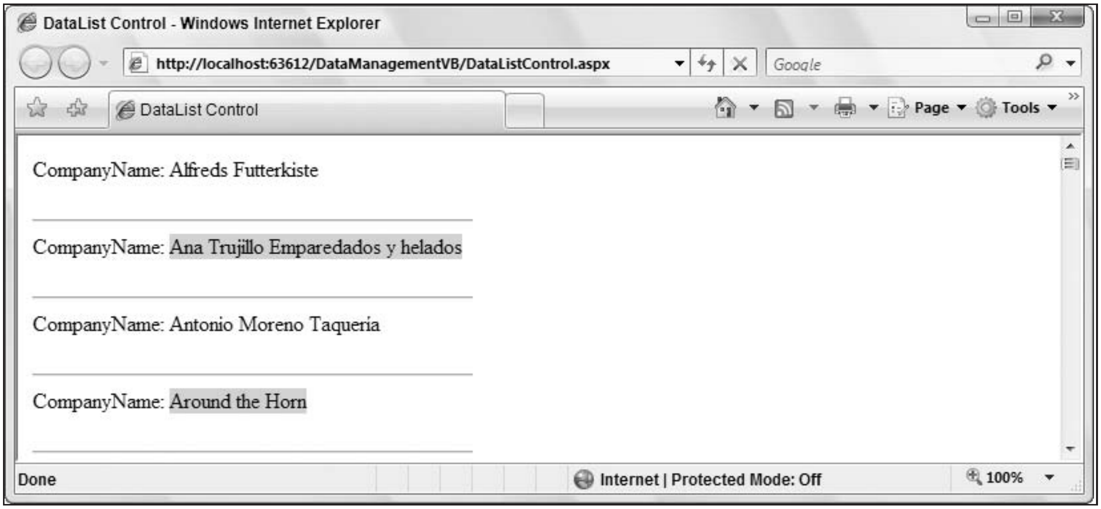
Working with Other Layout Templates

You will find that the other templates are just as easy to work with as the ItemTemplate. Listing 8-20 shows you how to add the AlternatingItemTemplate and the SeparatorTemplate to the company name display.

Listing 8-20: Using both the AlternatingItemTemplate and the SeparatorTemplate

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="DataListControl.aspx.vb"
    Inherits="DataListControl" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>DataList Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:DataList ID="DataList1" runat="server" DataSourceID="SqlDataSource1">
            <ItemTemplate>
                Company Name:
                <asp:Label ID="CompanyNameLabel" runat="server"
                    Text='<%# Eval("CompanyName") %>' />
                <br />
                <br />
            </ItemTemplate>
            <AlternatingItemTemplate>
                CompanyName:
                <asp:Label ID="CompanyNameLabel" runat="server"
                 BackColor="LightGray"
                 Text='<%# Eval("CompanyName") %>' />
                <br />
                <br />
            </AlternatingItemTemplate>
            <SeparatorTemplate>
                <hr />
            </SeparatorTemplate>
        </asp:DataList>
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:DSN_Northwind %>"
            SelectCommand="SELECT [CompanyName] FROM [Customers]">
        </asp:SqlDataSource>
    </div>
    </form>
</body>
</html>
```

In this case, the AlternatingItemTemplate is a repeat of the ItemTemplate, but the addition of the `BackColor` property to the Label control is contained within the item. The SeparatorTemplate is used between each item, whether it is from the ItemTemplate or the AlternatingItemTemplate. In this case, a simple `< hr / >` element is used to draw a line between each item. The output of this is shown here in Figure 8-6.


DataList Control - Windows Internet Explorer
http://localhost:63612/DataManagementVB/DataListControl.aspx
Google
DataList Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

CompanyName: Alfreds Futterkiste

CompanyName: Ana Trujillo Emparedados y helados

CompanyName: Antonio Moreno Taquería

CompanyName: Around the Horn

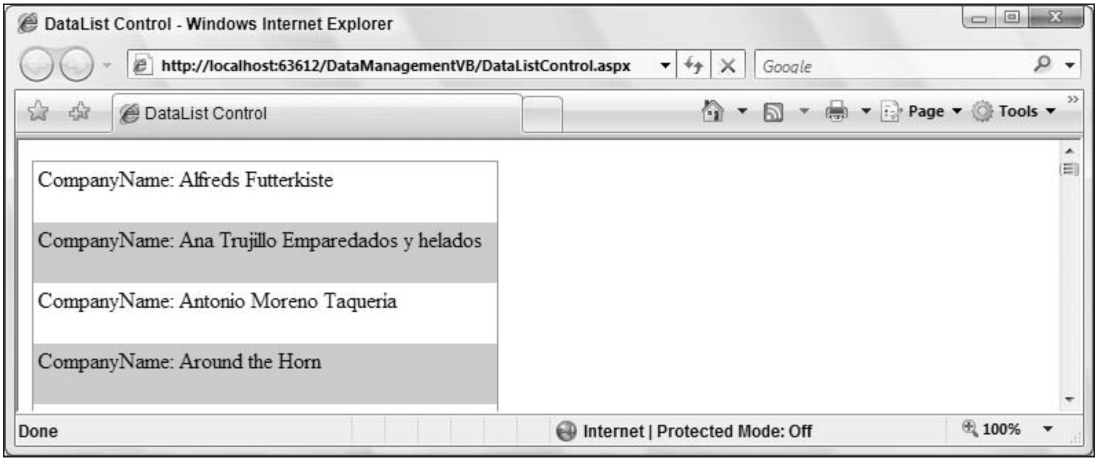
Figure 8-6

This process allows you to change how items are defined within the alternating rows and to put a separator between the elements. If you wanted just alternating row colors or an alternating style, it might not always be the best approach to use the `<AlternatingItemTemplate>` element, but you will find that it is better to use the `<AlternatingItemStyle>` element instead. This approach is presented here in Listing 8-21.

Listing 8-21: Using template styles

```
<asp:DataList ID="DataList1" runat="server" DataSourceID="SqlDataSource1"
 BackColor="White" BorderColor="#999999" BorderStyle="Solid" BorderWidth="1px"
 CellPadding="3" ForeColor="Black" GridLines="Vertical">
   <FooterStyle BackColor="#CCCCCC" />
   <AlternatingItemStyle BackColor="#CCCCCC" />
   <SelectedItemStyle BackColor="#000099" Font-Bold="True" ForeColor="White" />
   <HeaderStyle BackColor="Black" Font-Bold="True" ForeColor="White" />
   <ItemTemplate>
      CompanyName:
      <asp:Label ID="CompanyNameLabel" runat="server"
       Text='<%# Eval("CompanyName") %>' />
      <br />
      <br />
   </ItemTemplate>
</asp:DataList>
```

You will notice that each of the available templates also have an associated style element. Figure 8-7 shows the use of these styles.


DataList Control - Windows Internet Explorer
http://localhost:63612/DataManagementVB/DataListControl.aspx
Google
DataList Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

CompanyName: Alfreds Futterkiste

CompanyName: Ana Trujillo Emparedados y helados

CompanyName: Antonio Moreno Taqueria

CompanyName: Around the Horn

Figure 8-7

Working with Multiple Columns

Template-based controls are better at displaying items in multiple columns than other controls, such as the GridView control. The `RepeatColumns` property takes care of this. The code to make use of this property is shown in Listing 8-22.

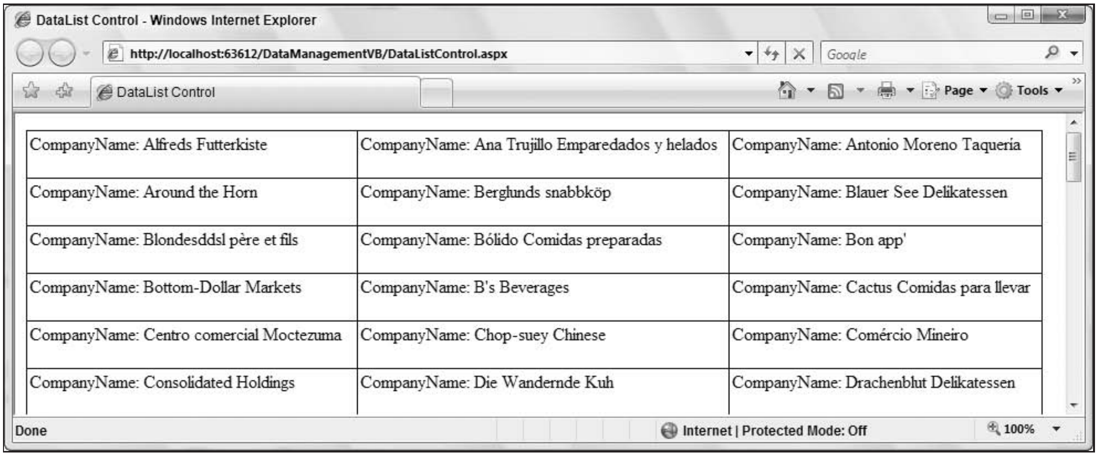
Listing 8-22: Creating multiple columns using the RepeatColumns property

```
<asp:DataList ID="DataList1" runat="server" DataSourceID="SqlDataSource1"
 CellPadding="2" RepeatColumns="3" RepeatDirection="Horizontal">
   <ItemTemplate>
    Company Name:
    <asp:Label ID="CompanyNameLabel" runat="server"
     Text='<%# Eval("CompanyName") %>' />
    <br />
    <br />
   </ItemTemplate>
</asp:DataList>
```

Running this bit of code in your page produces the results shown in Figure 8-8.

The `RepeatDirection` property instructs the DataList control about how to lay out the items bound to the control on the Web page. Possible values include `Vertical` and `Horizontal`. The default value is `Vertical`. Setting it to `Vertical` with a `RepeatColumn` setting of `3` gives the following results:

```
Item1    Item5   Item9
Item2    Item6   Item10
Item3    Item7   Item11
Item4    Item8   Item12
```


DataList Control - Windows Internet Explorer
http://localhost:63612/DataManagementVB/DataListControl.aspx
Google
DataList Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

CompanyName: Alfreds Futterkiste	CompanyName: Ana Trujillo Emparedados y helados	CompanyName: Antonio Moreno Taquería
CompanyName: Around the Horn	CompanyName: Berglunds snabbköp	CompanyName: Blauer See Delikatessen
CompanyName: Blondesddsl père et fils	CompanyName: Bólido Comidas preparadas	CompanyName: Bon app'
CompanyName: Bottom-Dollar Markets	CompanyName: B's Beverages	CompanyName: Cactus Comidas para llevar
CompanyName: Centro comercial Moctezuma	CompanyName: Chop-suey Chinese	CompanyName: Comércio Mineiro
CompanyName: Consolidated Holdings	CompanyName: Die Wandernde Kuh	CompanyName: Drachenblut Delikatessen

Figure 8-8

When the `RepeatDirection` property is set to `Horizontal`, you get the items laid out in a horizontal fashion:

```
Item1    Item2   Item3
Item4    Item5   Item6
Item7    Item8   Item9
Item10   Item11  Item12
```

The ListView Server Control

One of the newest template-based controls is the ListView control. This is a control that is only available in the 3.5 version of the .NET Framework. This control is considered a better alternative to the DataList control. You will find that this control gives you more control over the layout and works quite nicely in Visual Studio because it provides a set of wizards to easily set up your layout with the most common options.

Looking at the Available Templates

As with the DataList control, the ListView control has a series of available templates at your disposal. Each one of these templates controls a specific aspect of the layout. The following table defines the layout options available to this control.

Template	Description
`LayoutTemplate`	The core template that allows you to define the structure of the entire layout. Using this layout, you can use tables, spans, or anything else you want to layout your data elements.
`ItemTemplate`	Defines the layout for each individual item in the data collection.
`ItemSeparatorTemplate`	Defines the layout of any separator that is used between items.

Template	Description
`GroupTemplate`	A group container element that can contain any number of data items.
`GroupSeparatorTemplate`	Defines the layout of any separator that is used between groups.
`EmptyItemTemplate`	Defines the layout of the empty items that might be contained within a group. For instance, if you group by ten items and the last page contains only seven items, then the last three items will use this template.
`EmptyDataTemplate`	Defines the layout for items that do not contain data.
`SelectedItemTemplate`	Allows for a row or item to be defined on how it looks and behaves when selected.
`AlternatingItemTemplate`	Works in conjunction with the `ItemTemplate` to provide a layout for all the odd rows within the layout. This is commonly used if you want to have a grid or layout where each row is distinguished in some way (such as having a different background color).
`EditItemTemplate`	Allows for a row or item to be defined on how it looks and behaves when editing.
`InsertItemTemplate`	Allows for a row or item to be defined on how it looks and behaves when performing an insert.

Next, the following sections look at using some of these in your ASP.NET page.

Using the Templates

In creating a page that makes use of the ListView control, the first step will be to create a basic page with a ListView control on it, as illustrated here in Listing 8-23.

Listing 8-23: Creating the base page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="ListViewControl.aspx.vb"
    Inherits="ListViewControl" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>ListView Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:ListView ID="ListView1" runat="server" DataKeyNames="CustomerID"
            DataSourceID="SqlDataSource1">
        </asp:ListView>
```

Continued

```
            <asp:SqlDataSource ID="SqlDataSource1" runat="server"
                ConnectionString="<%$ ConnectionStrings:DSN_Northwind %>"
                SelectCommand="SELECT * FROM [Customers] ORDER BY [CompanyName]"
                InsertCommand="INSERT INTO [Customers] ([CustomerID], [CompanyName],
                   [ContactName], [ContactTitle], [Address], [City], [Region],
                   [PostalCode], [Country], [Phone], [Fax]) VALUES (@CustomerID,
                   @CompanyName, @ContactName, @ContactTitle, @Address, @City, @Region,
                   @PostalCode, @Country, @Phone, @Fax)"
                UpdateCommand="UPDATE [Customers] SET [CompanyName] = @CompanyName,
                   [ContactName] = @ContactName, [ContactTitle] = @ContactTitle,
                   [Address] = @Address, [City] = @City, [Region] = @Region,
                   [PostalCode] = @PostalCode, [Country] = @Country, [Phone] = @Phone,
                   [Fax] = @Fax WHERE [CustomerID] = @CustomerID">
                <UpdateParameters>
                    <asp:Parameter Name="CompanyName" Type="String" />
                    <asp:Parameter Name="ContactName" Type="String" />
                    <asp:Parameter Name="ContactTitle" Type="String" />
                    <asp:Parameter Name="Address" Type="String" />
                    <asp:Parameter Name="City" Type="String" />
                    <asp:Parameter Name="Region" Type="String" />
                    <asp:Parameter Name="PostalCode" Type="String" />
                    <asp:Parameter Name="Country" Type="String" />
                    <asp:Parameter Name="Phone" Type="String" />
                    <asp:Parameter Name="Fax" Type="String" />
                    <asp:Parameter Name="CustomerID" Type="String" />
                </UpdateParameters>
                <InsertParameters>
                    <asp:Parameter Name="CustomerID" Type="String" />
                    <asp:Parameter Name="CompanyName" Type="String" />
                    <asp:Parameter Name="ContactName" Type="String" />
                    <asp:Parameter Name="ContactTitle" Type="String" />
                    <asp:Parameter Name="Address" Type="String" />
                    <asp:Parameter Name="City" Type="String" />
                    <asp:Parameter Name="Region" Type="String" />
                    <asp:Parameter Name="PostalCode" Type="String" />
                    <asp:Parameter Name="Country" Type="String" />
                    <asp:Parameter Name="Phone" Type="String" />
                    <asp:Parameter Name="Fax" Type="String" />
                </InsertParameters>
            </asp:SqlDataSource>
        </div>
        </form>
    </body>
    </html>
```

In this case, you have a base ListView control and a SqlDataSource control that has been wired up to the Northwind sample database and provided Select, Update, and Insert methods. The ListView control itself is then bound to the SqlDataSource control. It provides the primary key of the table for working with the various queries through the use of the `DataKeyNames` property.

Creating the Layout Template

The next step is to create the layout of the overall control using the LayoutTemplate. The use of this template is illustrated in Listing 8-24.

Listing 8-24: Using the LayoutTemplate element

```
<LayoutTemplate>
   <table runat="server">
      <tr runat="server">
         <td runat="server">
            <table ID="itemPlaceholderContainer" runat="server" border="1"
             style="background-color: #FFFFFF;border-collapse: collapse;
                    border-color: #999999;border-style:none;border-width:1px;
                    font-family: Verdana, Arial, Helvetica, sans-serif;">
              <tr runat="server" style="background-color:#DCDCDC;color: #000000;">
                 <th runat="server"></th>
                 <th runat="server">Customer ID</th>
                 <th runat="server">Company Name</th>
                 <th runat="server">Contact Name</th>
              </tr>
              <tr ID="itemPlaceholder" runat="server"></tr>
            </table>
         </td>
      </tr>
      <tr runat="server">
         <td runat="server" style="text-align: center;background-color: #CCCCCC;
          font-family: Verdana, Arial, Helvetica, sans-serif;color: #000000;">
            <asp:DataPager ID="DataPager1" runat="server">
               <Fields>
                  <asp:NextPreviousPagerField ButtonType="Button"
                   ShowFirstPageButton="True" ShowNextPageButton="False"
                   ShowPreviousPageButton="False" />
                  <asp:NumericPagerField />
                  <asp:NextPreviousPagerField ButtonType="Button"
                   ShowLastPageButton="True" ShowNextPageButton="False"
                   ShowPreviousPageButton="False" />
               </Fields>
            </asp:DataPager>
         </td>
      </tr>
   </table>
</LayoutTemplate>
```

This layout template constructs the layout as a grid using tables to layout the items. A styled table is defined with a header in place. The most important part of laying out the template is that the container itself is defined using a control with the `ID` value of `itemPlaceholderContainer`. The element will also need to be made a server control by adding the `runat` property.

```
<table ID="itemPlaceholderContainer" runat="server" border="1"
 style="background-color: #FFFFFF;border-collapse: collapse;
 border-color: #999999;border-style:none;border-width:1px;
 font-family: Verdana, Arial, Helvetica, sans-serif;">

</table>
```

The placeholder for each data item needs to take the same form, but the `ID` of the server control you make needs to have a value of `itemPlaceholder`.

```
<table ID="itemPlaceholderContainer" runat="server" border="1"
 style="background-color: #FFFFFF;border-collapse: collapse;
 border-color: #999999;border-style:none;border-width:1px;
 font-family: Verdana, Arial, Helvetica, sans-serif;">
   <tr runat="server" style="background-color:#DCDCDC;color: #000000;">
      <th runat="server"></th>
      <th runat="server">Customer ID</th>
      <th runat="server">Company Name</th>
      <th runat="server">Contact Name</th>
   </tr>
   <tr ID="itemPlaceholder" runat="server"></tr>
</table>
```

It is important to keep the `itemPlaceholder` element within the itemPlaceholderContainer control, within the layout template. It cannot sit outside of the container.

The final part of this layout is the new DataPager server control. This new server control is part of ASP.NET 3.5.

```
<asp:DataPager ID="DataPager1" runat="server">
   <Fields>
      <asp:NextPreviousPagerField ButtonType="Button"
       ShowFirstPageButton="True" ShowNextPageButton="False"
       ShowPreviousPageButton="False" />
      <asp:NumericPagerField />
      <asp:NextPreviousPagerField ButtonType="Button"
       ShowLastPageButton="True" ShowNextPageButton="False"
       ShowPreviousPageButton="False" />
   </Fields>
</asp:DataPager>
```

The DataPager works with template-based data in allowing you to control how end users move across the pages of the data collection.

Now that the LayoutTemplate is in place, the next step is to create the ItemTemplate.

Creating the ItemTemplate

The ItemTemplate that you create is quite similar to the ItemTemplate that is part of the DataList control that was discussed earlier. In this case, however, the ItemTemplate is placed in the specific spot within the layout of the page where you defined the itemPlaceholder control to be. Listing 8-25 shows the ItemTemplate for this example.

Listing 8-25: Building the ItemTemplate

```
<ItemTemplate>
   <tr style="background-color:#DCDCDC;color: #000000;">
      <td>
         <asp:Button ID="EditButton" runat="server"
          CommandName="Edit" Text="Edit" />
      </td>
      <td>
```

```
            <asp:Label ID="CustomerIDLabel" runat="server"
             Text='<%# Eval("CustomerID") %>' />
         </td>
         <td>
            <asp:Label ID="CompanyNameLabel" runat="server"
             Text='<%# Eval("CompanyName") %>' />
         </td>
         <td>
            <asp:Label ID="ContactNameLabel" runat="server"
             Text='<%# Eval("ContactName") %>' />
         </td>
      </tr>
   </ItemTemplate>
```

Creating the EditItemTemplate

The EditItemTemplate is the area that shows up when you decide to edit the data item (in this case, a row of data). Listing 8-26 shows the EditItemTemplate in use.

Listing 8-26: Building the EditItemTemplate

```
<EditItemTemplate>
   <tr style="background-color:#008A8C;color: #FFFFFF;">
      <td>
         <asp:Button ID="UpdateButton" runat="server"
          CommandName="Update" Text="Update" />
         <asp:Button ID="CancelButton" runat="server"
          CommandName="Cancel" Text="Cancel" />
      </td>
      <td>
         <asp:Label ID="CustomerIDLabel1" runat="server"
          Text='<%# Eval("CustomerID") %>' />
      </td>
      <td>
         <asp:TextBox ID="CompanyNameTextBox" runat="server"
          Text='<%# Bind("CompanyName") %>' />
      </td>
      <td>
         <asp:TextBox ID="ContactNameTextBox" runat="server"
          Text='<%# Bind("ContactName") %>' />
      </td>
   </tr>
</EditItemTemplate>
```

In this case, the EditItemTemplate, when shown, displays an Update and Cancel button to manipulate the editing options. When editing, the values are placed within text boxes and the values are then updated into the database through the `Update`command.

Creating the EmptyItemTemplate

If there are no values in the database, then you should prepare to gracefully show something in your layout. The `EmptyItemTemplate` is usedin Listing 8-27 to perform that operation.

Listing 8-27: Building the EmptyItemTemplate

```
<EmptyDataTemplate>
   <table runat="server"
    style="background-color: #FFFFFF;border-collapse: collapse;
        border-color: #999999;border-style:none;border-width:1px;">
      <tr>
         <td>No data was returned.</td>
      </tr>
   </table>
</EmptyDataTemplate>
```

Creating the InsertItemTemplate

The last section looked at here is the `InsertItemTemplate`. This section allows you to define how a form should be laid out for inserting data, similar to that used in the `ItemTemplate`, into the data store.

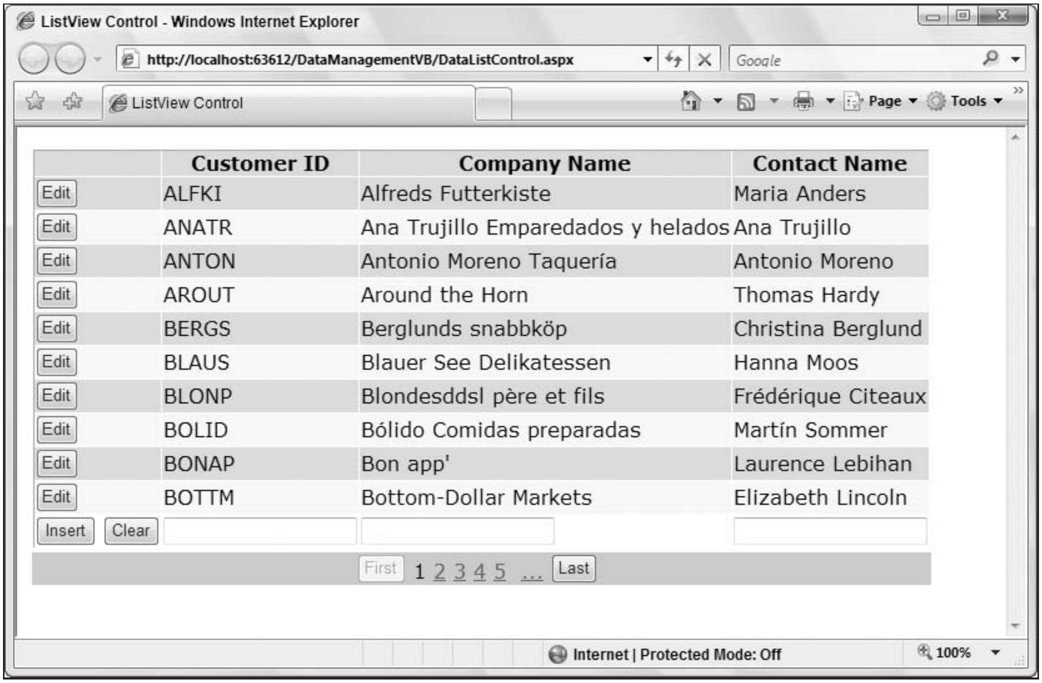
Listing 8-28 shows an example of the `InsertItemTemplate`.

Listing 8-28: Building the InsertItemTemplate

```
<InsertItemTemplate>
   <tr style="">
      <td>
         <asp:Button ID="InsertButton" runat="server" CommandName="Insert"
          Text="Insert" />
         <asp:Button ID="CancelButton" runat="server" CommandName="Cancel"
          Text="Clear" />
      </td>
      <td>
         <asp:TextBox ID="CustomerIDTextBox" runat="server"
          Text='<%# Bind("CustomerID") %>' />
      </td>
      <td>
         <asp:TextBox ID="CompanyNameTextBox" runat="server"
          Text='<%# Bind("CompanyName") %>' />
      </td>
      <td>
         <asp:TextBox ID="ContactNameTextBox" runat="server"
          Text='<%# Bind("ContactName") %>' />
      </td>
   </tr>
</InsertItemTemplate>
```

The Results

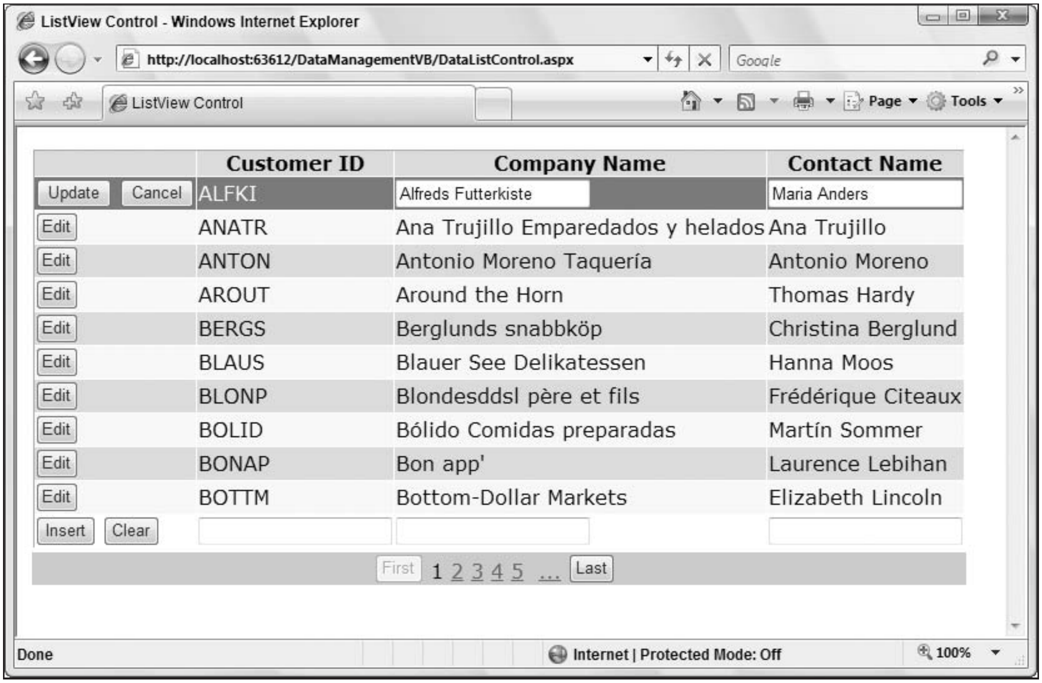
After you have created an additional AlternatingItemTemplate that is the same as the ItemTemplate (but styled differently), you can then run the page. Then you will be presented with your own custom grid. An example is presented in Figure 8-9.


ListView Control - Windows Internet Explorer
http://localhost:63612/DataManagementVB/DataListControl.aspx
Google
ListView Control
Page
Tools
Internet | Protected Mode: Off
100%

	Customer ID	Company Name	Contact Name
Edit	ALFKI	Alfreds Futterkiste	Maria Anders
Edit	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo
Edit	ANTON	Antonio Moreno Taquería	Antonio Moreno
Edit	AROUT	Around the Horn	Thomas Hardy
Edit	BERGS	Berglunds snabbköp	Christina Berglund
Edit	BLAUS	Blauer See Delikatessen	Hanna Moos
Edit	BLONP	Blondesddsl père et fils	Frédérique Citeaux
Edit	BOLID	Bólido Comidas preparadas	Martín Sommer
Edit	BONAP	Bon app'	Laurence Lebihan
Edit	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln
Insert Clear			

First 1 2 3 4 5 ... Last

Figure 8-9


ListView Control - Windows Internet Explorer
http://localhost:63612/DataManagementVB/DataListControl.aspx
Google
ListView Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

	Customer ID	Company Name	Contact Name
Update Cancel	ALFKI	Alfreds Futterkiste	Maria Anders
Edit	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo
Edit	ANTON	Antonio Moreno Taquería	Antonio Moreno
Edit	AROUT	Around the Horn	Thomas Hardy
Edit	BERGS	Berglunds snabbköp	Christina Berglund
Edit	BLAUS	Blauer See Delikatessen	Hanna Moos
Edit	BLONP	Blondesddsl père et fils	Frédérique Citeaux
Edit	BOLID	Bólido Comidas preparadas	Martín Sommer
Edit	BONAP	Bon app'	Laurence Lebihan
Edit	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln
Insert Clear			

First 1 2 3 4 5 ... Last

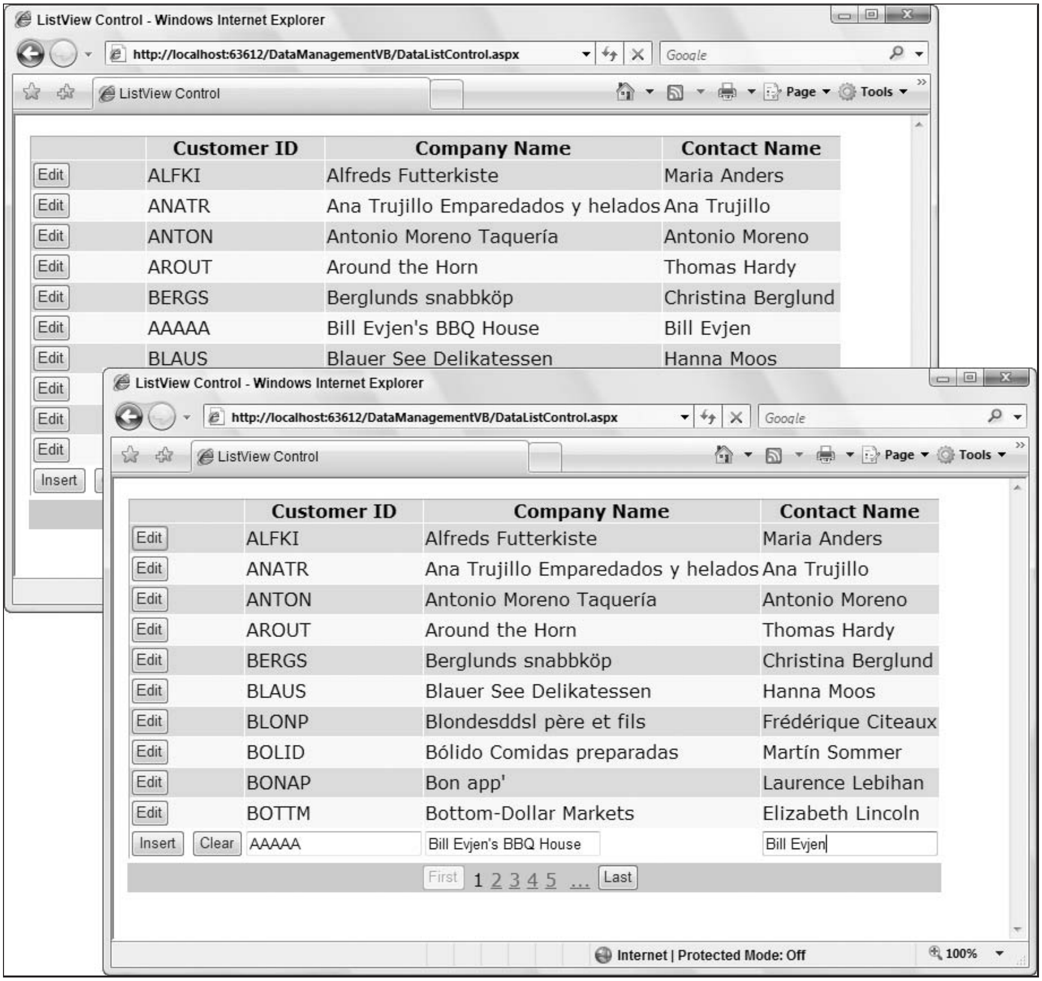
Figure 8-10

From this figure, you can see that all your defined elements are in place. The header is defined through the use of the LayoutTemplate. The items in the grid are defined through the use of the ItemTemplate. The AlternatingItemTemplate, the insert form, is defined through the use of the InsertTemplate. The page navigation is defined by the new DataPager server control. Again, the DataPager control is defined within the LayoutTemplate itself.

Editing items in this template is as simple as clicking on the Edit button. This will change the view to the EditTemplate for the selected item, as illustrated in Figure 8-10.

Once you enter the edit mode here, you can change any of the values within the text boxes and then click the Update button to update the data to the new values. You can also cancel out of the operation by clicking the Cancel button.

Inserting data is as simple as filling out the form and clicking on the Insert button, as illustrated in Figure 8-11.


ListView Control - Windows Internet Explorer
http://localhost:63612/DataManagementVB/DataListControl.aspx
Google
ListView Control
Page
Tools
Internet | Protected Mode: Off
100%

	Customer ID	Company Name	Contact Name
Edit	ALFKI	Alfreds Futterkiste	Maria Anders
Edit	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo
Edit	ANTON	Antonio Moreno Taquería	Antonio Moreno
Edit	AROUT	Around the Horn	Thomas Hardy
Edit	BERGS	Berglunds snabbköp	Christina Berglund
Edit	AAAAA	Bill Evjen's BBQ House	Bill Evjen
Edit	BLAUS	Blauer See Delikatessen	Hanna Moos
Edit			
Edit			
Edit			
Insert			

	Customer ID	Company Name	Contact Name
Edit	ALFKI	Alfreds Futterkiste	Maria Anders
Edit	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo
Edit	ANTON	Antonio Moreno Taquería	Antonio Moreno
Edit	AROUT	Around the Horn	Thomas Hardy
Edit	BERGS	Berglunds snabbköp	Christina Berglund
Edit	BLAUS	Blauer See Delikatessen	Hanna Moos
Edit	BLONP	Blondesddsl père et fils	Frédérique Citeaux
Edit	BOLID	Bólido Comidas preparadas	Martín Sommer
Edit	BONAP	Bon app'	Laurence Lebihan
Edit	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln
Insert Clear	AAAAA	Bill Evjen's BBQ House	Bill Evjen

First 1 2 3 4 5 ... Last

Figure 8-11

Although this example shows a grid as the output of the new ListView control, you can also structure it so that your data items are presented in any fashion you want (such as bulleted lists).

Using Visual Studio for ADO.NET Tasks

Earlier, this chapter covered how to construct a DataSet and how to fill it with data using the DataAdapter. Although you can always build this construction yourself, you also have the option of building data access into your ASP.NET applications using some of the wizards available from Visual Studio 2008.

The following example, which is a little bit of a lengthy one, shows you how to build an ASP.NET page that displays the results from a DataSet that gets its data from two separate tables. You will discover several different wizards in Visual Studio that you can work with when using ADO.NET.

Creating a Connection to the Data Source

As in code, one of the first things you do when working with data is make a connection to the data source. Visual Studio provides a visual way to make connections to your data stores. In this case, you will want to make a connection to the Northwind database in SQL Server.

When you open the Server Explorer, you will notice a section for data connections (see Figure 8-12).

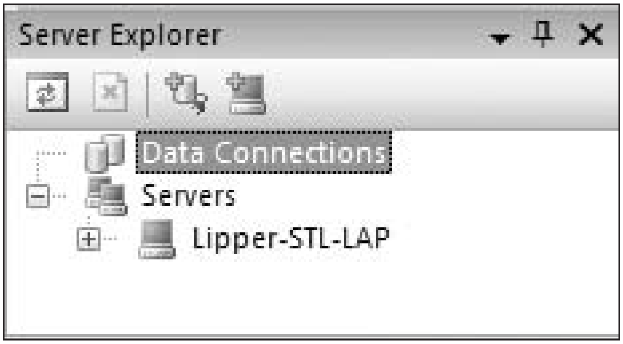
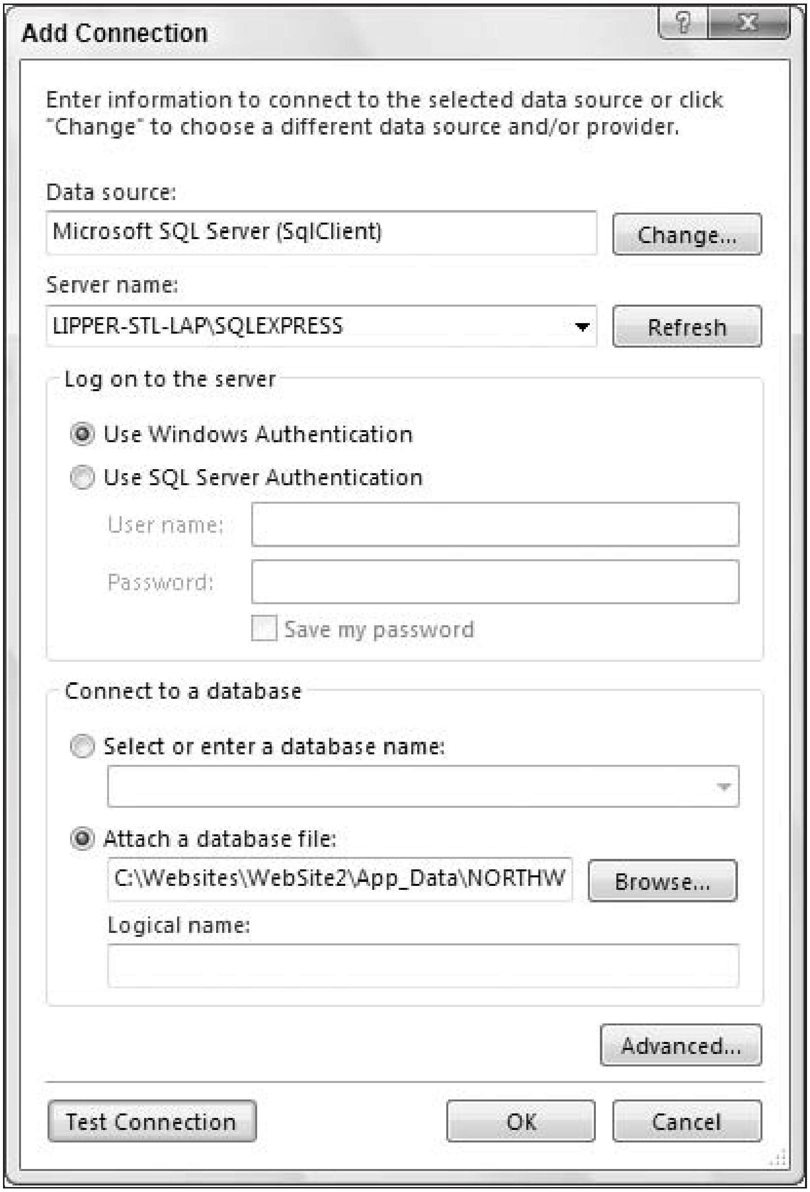


Figure 8-12

The steps to create a data connection to the Northwind database in SQL Server are straightforward. Right-click on Data Connections and choose Add Connection. You are presented with the Data Link Properties dialog box. This dialog box, by default, asks for a connection to SQL Server. If you are going to connect to a different source, such as Microsoft Access, simply click on the Provider tab and change the provider.

Figure 8-13 shows the Add Connection dialog box and the settings that you need in order to connect to your local SQL Server Express Edition.

If you are connecting to a SQL Server that resides on your localhost, you want to put a period (.) in the box that asks you to select or enter a server name. If you are working from a local SQL Server Express Edition file in your project (such as what is shown here in Figure 8-13), then you are going to want to use your server name with `\SQLEXPRESS`. Put in your login credentials for SQL Server and then select the database that you wish to make the connection to by using the drop-down list. The other option, if you are using a SQL Server Express Edition file, is to select the physical database file by using the Attach a Database File option.



Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient)

Server name: LIPPER-STL-LAP\SQLEXPRESS

Log on to the server

● Use Windows Authentication

○ Use SQL Server Authentication

User name:

Password:

☐ Save my password

Connect to a database

○ Select or enter a database name:

● Attach a database file: C:\Websites\WebSite2\App_Data\NORTHW

Logical name:

Figure 8-13

From this dialog box, you can also test the connection to ensure that everything works properly. If everything is in place, you get a confirmation stating such. Clicking OK will then cause a connection to appear in the Solution Explorer.

Expanding this connection, you find a way to access the data source just as you would by using the SQL Server Enterprise Manager (see Figure 8-14).

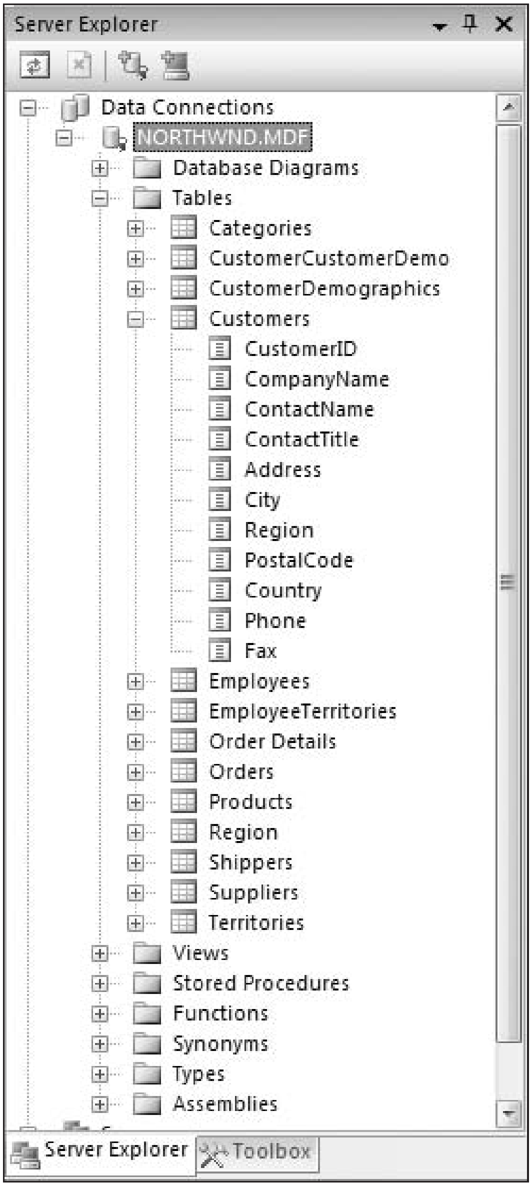

Server Explorer
Data Connections
NORTHWND.MDF
Database Diagrams
Tables
Categories
CustomerCustomerDemo
CustomerDemographics
Customers
CustomerID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax
Employees
EmployeeTerritories
Order Details
Orders
Products
Region
Shippers
Suppliers
Territories
Views
Stored Procedures
Functions
Synonyms
Types
Assemblies
Server Explorer
Toolbox

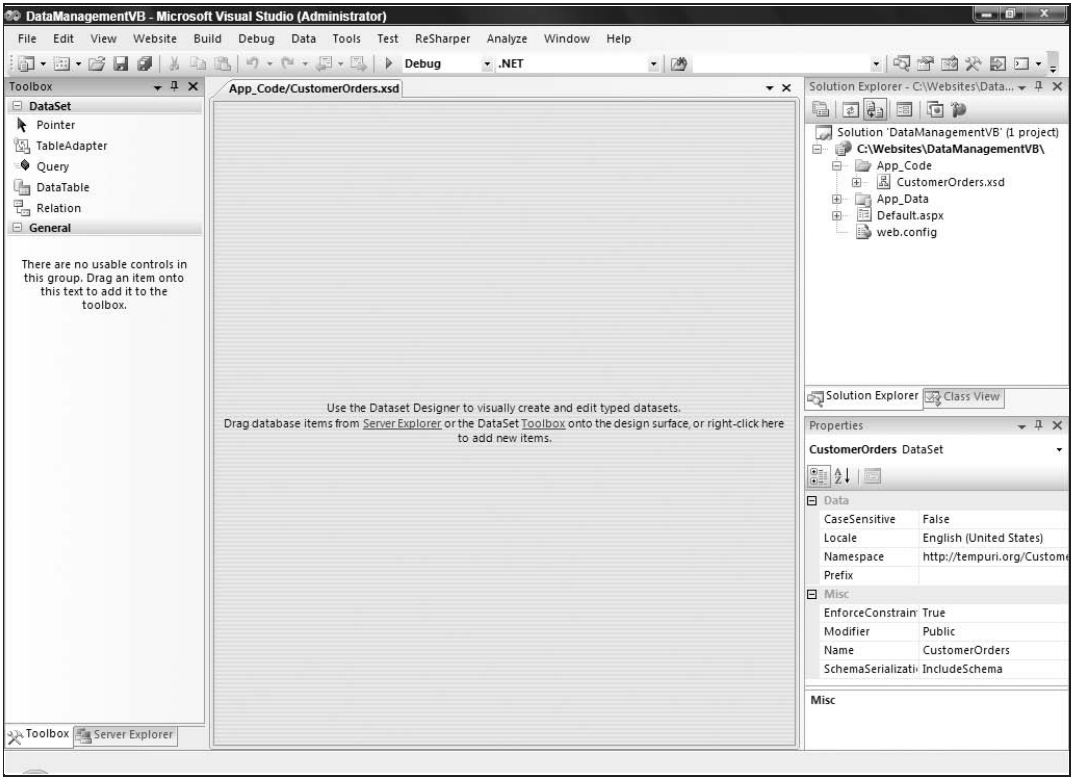
Figure 8-14

From here, you can work with the database and view information about all the tables and fields that are contained within the database. More specifically, you can view and work with Database Diagrams, Tables, Views, Stored Procedures, and Functions.

After you have run through this wizard, you have a connection to the Northwind database that can be used by any components that you place on any component designer that you might be working with in your application.

Working with a Dataset Designer

The next step is to create a typed `DataSet` object in your project that pulls its data from the Northwind database. First you need to make sure that your application has an `App_Code` folder within the solution. Right-clicking on the folder will allow you to add a new item to the folder. From the provided dialog box, add a DataSet called `CustomerOrders.xsd`. You will then be presented with the message shown in Figure 8-15.


DataManagementVB - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Data Tools Test ReSharper Analyze Window Help
Debug
.NET
Toolbox
DataSet
Pointer
TableAdapter
Query
DataTable
Relation
General
App_Code/CustomerOrders.xsd
Solution Explorer - C:\Websites\Data...
Solution 'DataManagementVB' (1 project)
C:\Websites\DataManagementVB\
App_Code
CustomerOrders.xsd
App_Data
Default.aspx
web.config
Solution Explorer
Class View
Toolbox
Server Explorer
Properties

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

Use the Dataset Designer to visually create and edit typed datasets.
Drag database items from Server Explorer or the DataSet Toolbox onto the design surface, or right-click here to add new items.

CustomerOrders DataSet

Data	
CaseSensitive	False
Locale	English (United States)
Namespace	http://tempuri.org/Custome
Prefix	
Misc	
EnforceConstrain	True
Modifier	Public
Name	CustomerOrders
SchemaSerializati	IncludeSchema

Misc

Figure 8-15

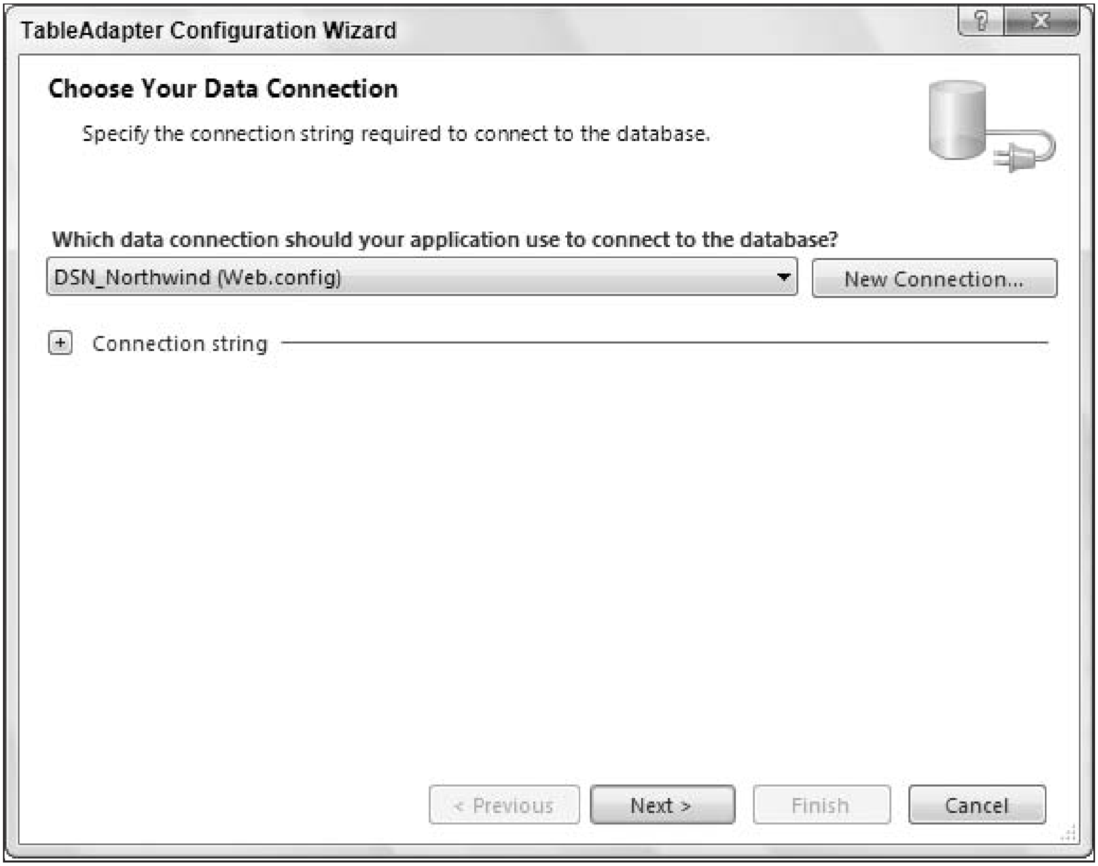
This page is referred to as the Dataset Designer. This is the design surface for any non-visual components that you incorporate within your `DataSet` object. Just as you can drag and drop controls onto a design surface for any Windows Forms or Web Forms application, the Dataset Designer enables you to drag and drop components onto this surface.

A component does not appear visually in your applications, but a visual representation of the component sits on the design surface. Highlighting the component allows you to modify its settings and properties in the Properties window.

What can you drag and drop onto this surface? In the following examples, you see how to work with `TableAdapter` and `DataTable` objects on this design surface. If you open up the Toolbox window, and click the DataSet tab, you see some additional components that can be used on this design surface.

The goal of this example is to return a DataSet to the end user through an XML Web service. To accomplish this, you have to incorporate a `DataAdapter` to extract the data from the data source and to populate the `DataSet` before passing it on.

This example uses the Northwind database and the first step you need to take is to drag and drop a `TableAdapter` onto the Dataset design surface. Dragging and dropping a `TableAdapter` onto your design surface causes a wizard to appear, as shown in Figure 8-16.



Choose Your Data Connection

Specify the connection string required to connect to the database.

Which data connection should your application use to connect to the database?

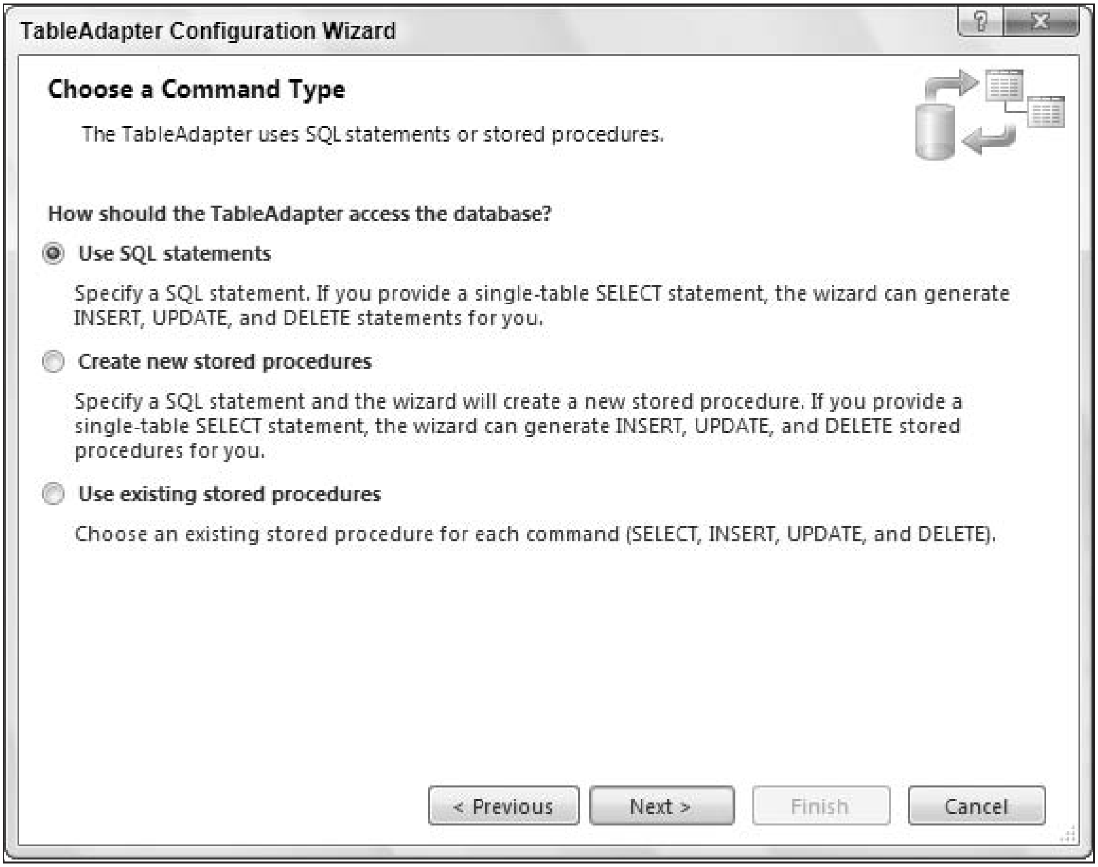
DSN_Northwind (Web.config)

Figure 8-16

Because you want this DataSet to contain two DataTables — one for the Customers table and another for the Orders table — you have to go through this process twice.

It is important to note that the job of the `TableAdapter` object is to make the connection to the specified table as well as to perform all the select, update, insert, and delete commands that are required. For this example, you simply want the `TableAdapter` to make the select call and then later to update any changes that are made back to the SQL Server.

As you work through the wizard, you come to a screen that asks how you want to query the database (see Figure 8-17). You have three options: using SQL statements, using stored procedures that have already been created, or building brand-new stored procedures directly from this wizard.



Choose a Command Type

The TableAdapter uses SQL statements or stored procedures.

How should the TableAdapter access the database?

● Use SQL statements

Specify a SQL statement. If you provide a single-table SELECT statement, the wizard can generate INSERT, UPDATE, and DELETE statements for you.

○ Create new stored procedures

Specify a SQL statement and the wizard will create a new stored procedure. If you provide a single-table SELECT statement, the wizard can generate INSERT, UPDATE, and DELETE stored procedures for you.

○ Use existing stored procedures

Choose an existing stored procedure for each command (SELECT, INSERT, UPDATE, and DELETE).

Figure 8-17

For this example, choose Use SQL statements. Selecting this option brings you to a text box where you can write your own SQL statement if you wish.

The great thing about this process is that, after you create a SQL `select` command, the TableAdapter wizard also creates the associated `insert`, `update`, and `delete` commands for you. You also have the option of building your queries using the Query Builder. This enables you to graphically design the query yourself. If this option is selected, you can choose from a list of tables in the Northwind database. For the first TableAdapter, choose Customers. For the second TableAdapter choose Orders. You make your selection by clicking the Add button and then closing the dialog box (see Figure 8-18).

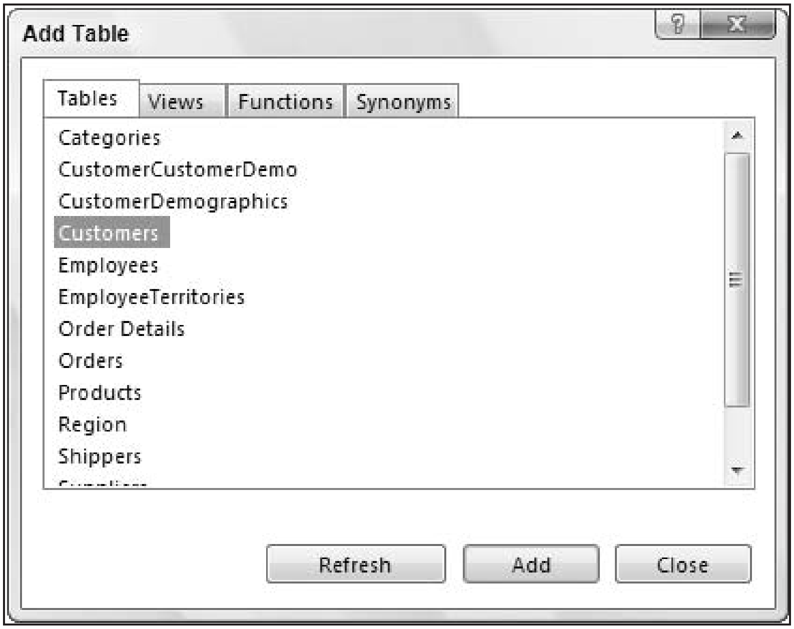
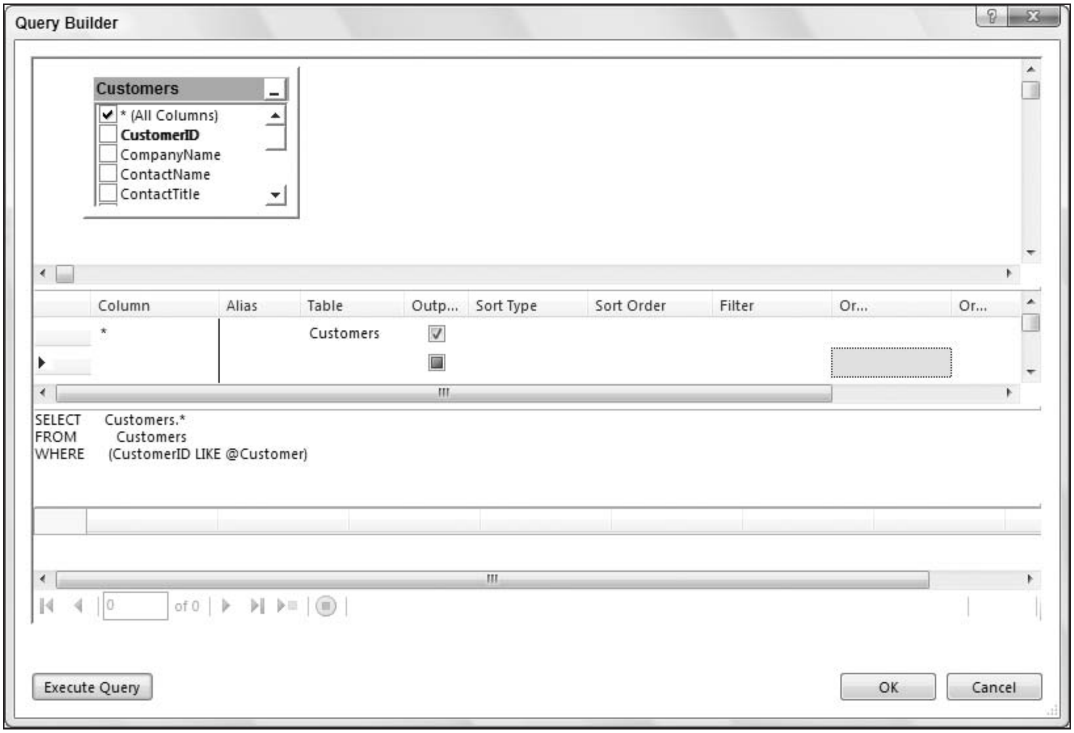

Add Table
Tables
Views
Functions
Synonyms
Categories
CustomerCustomerDemo
CustomerDemographics
Customers
Employees
EmployeeTerritories
Order Details
Orders
Products
Region
Shippers
Refresh
Add
Close

Figure 8-18


Query Builder
Customers
* (All Columns)
CustomerID
CompanyName
ContactName
ContactTitle
Column
Alias
Table
Outp...
Sort Type
Sort Order
Filter
Or...
Or...
*
Customers
0
of 0
Execute Query
OK
Cancel

```
SELECT    Customers.*
FROM        Customers
WHERE     (CustomerID LIKE @Customer)
```

Figure 8-19

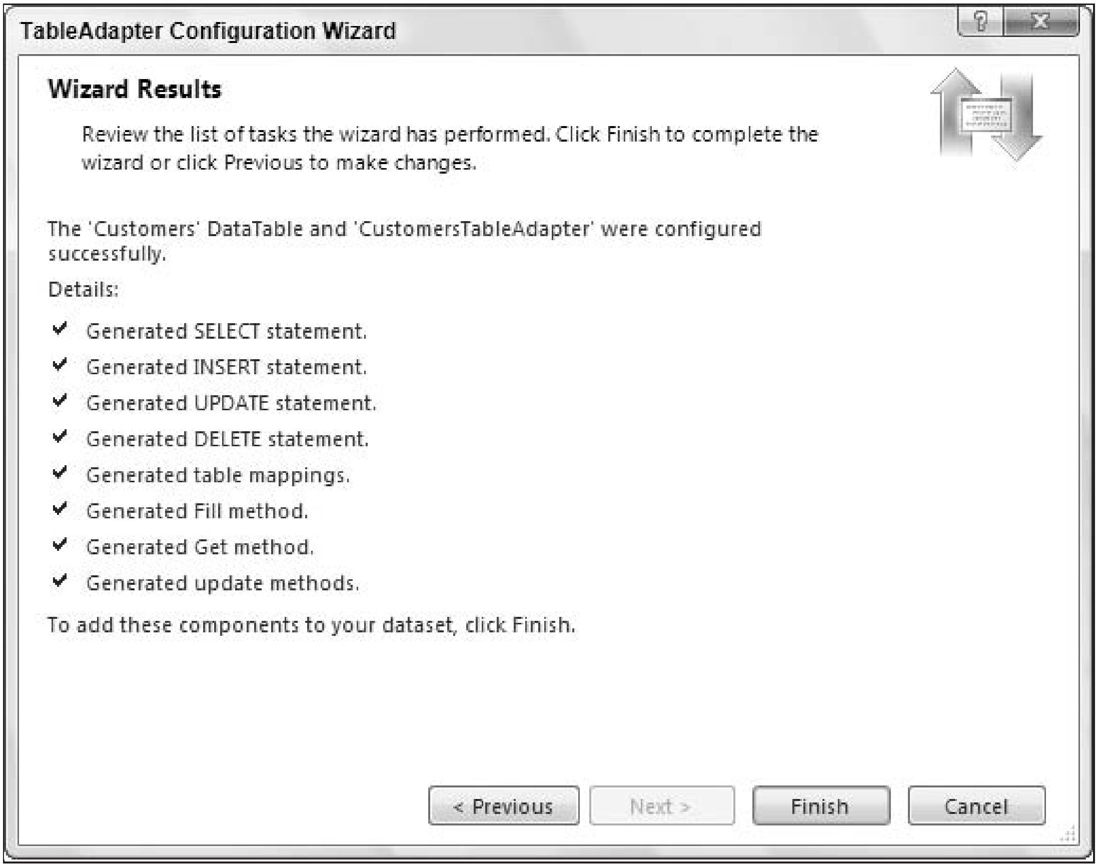
After you close the Add Table dialog box, you see a visual representation of the table that you selected in the Query Builder dialog box (see Figure 8-19). You can then select some or all the fields to be returned from the query. For this example, you want everything returned from both the Customers and the Orders table, so select the first check box with the asterisk (*). Notice that the query listed in this dialog box now says `SELECT * FROM Customers`. After the word ''Customers,'' add text to the query so that it looks like the following:

```
SELECT Customers.* FROM Customers WHERE (CustomerID LIKE @Customer)
```

With this query, you specify that you want to return the customer information when the `CustomerID` fits the parameter that you pass into the query from your code (using `@Customer`).

After your query is in place, simply click OK and then click the Next button to have not only the `select` query, but also the `insert`, `update`, and `delete` queries generated for you.

Figure 8-20 shows you the final page after all the queries have been generated.



Wizard Results

Review the list of tasks the wizard has performed. Click Finish to complete the wizard or click Previous to make changes.

The 'Customers' DataTable and 'CustomersTableAdapter' were configured successfully.

Details:

- ✓ Generated SELECT statement.
- ✓ Generated INSERT statement.
- ✓ Generated UPDATE statement.
- ✓ Generated DELETE statement.
- ✓ Generated table mappings.
- ✓ Generated Fill method.
- ✓ Generated Get method.
- ✓ Generated update methods.

To add these components to your dataset, click Finish.

Figure 8-20

After you reach this point, you can either click the Previous button to return to one of the prior steps in order to change a setting or the query itself, or you can click the Finish button to apply everything to your TableAdapter. After you are finished using the wizard, notice there is a visual representation of the `CustomersTableAdapter` that you just created (see Figure 8-21). Along with that is a `DataTable` object for the Customers table. The `TableAdapter` and the `DataTable` objects that are shown on the design surface are also labeled with their IDs. Therefore, in your code, you can address this TableAdapter that you just built by referring to it as `CustomerOrdersTableAdapters.CustomersTableAdapter`. The second TableAdapter that queries the Orders table is then shown and referred to as `CustomerOrdersTableAdapters.OrdersTableAdapter`.

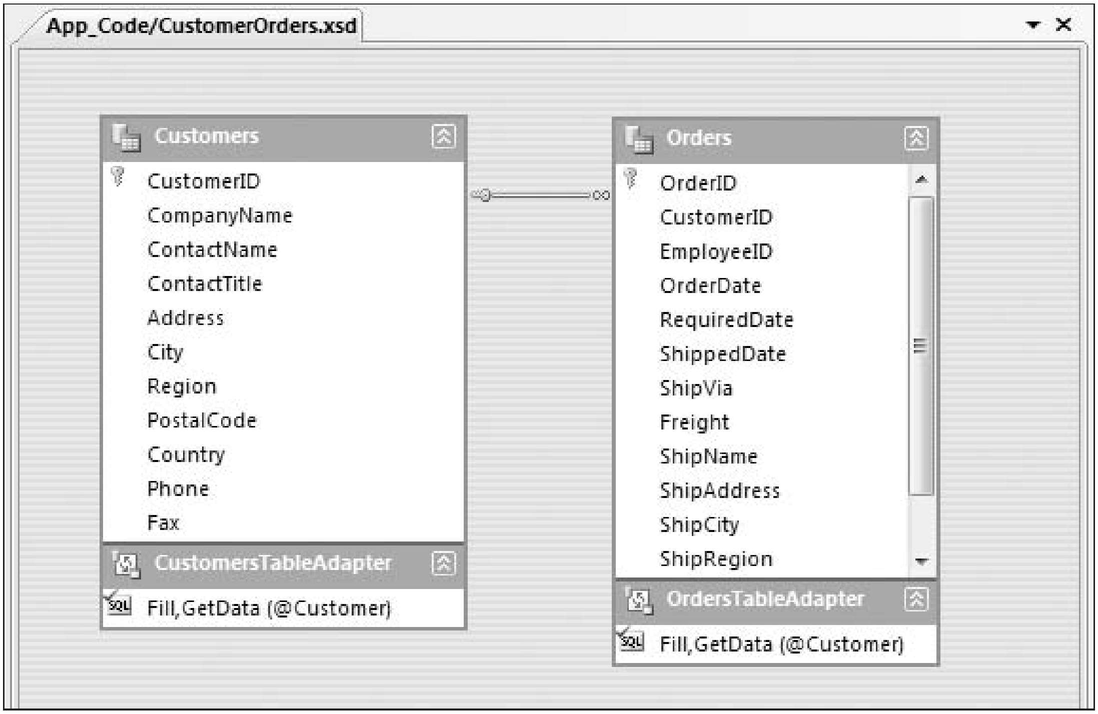

App_Code/CustomerOrders.xsd
Customers
CustomerID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax
CustomersTableAdapter
Fill,GetData (@Customer)
Orders
OrderID
CustomerID
EmployeeID
OrderDate
RequiredDate
ShippedDate
ShipVia
Freight
ShipName
ShipAddress
ShipCity
ShipRegion
OrdersTableAdapter
Fill,GetData (@Customer)

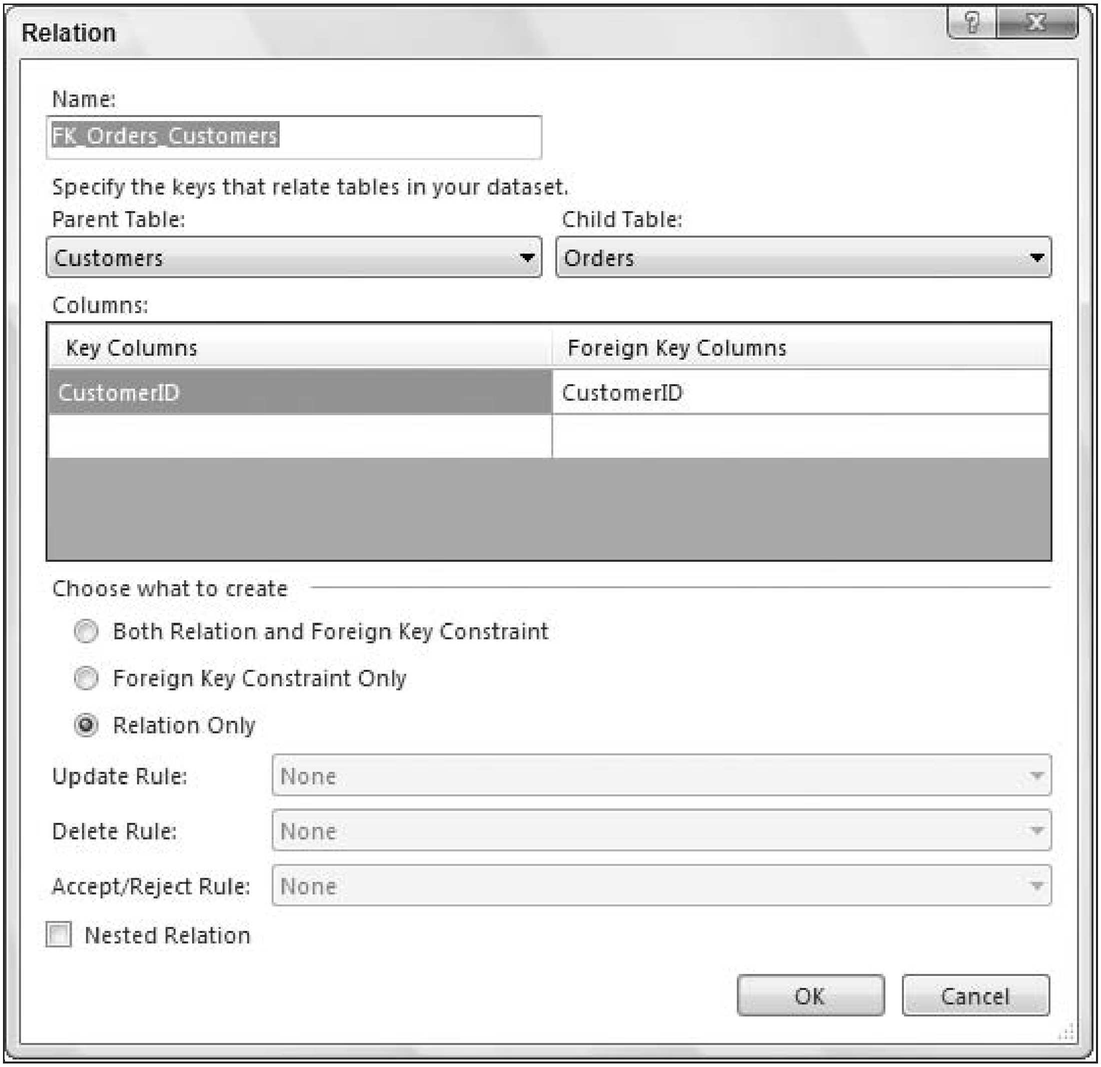
Figure 8-21

After you have the two DataAdapters in place, you will also notice that there is an automatic relation put into place for you. This is represented by the line between the two items on the page. Right-clicking on the relation, you can edit the relation with the Relation dialog box (see Figure 8-22).

In the end, Visual Studio has taken care of a lot for you. Again, this is not the only way to complete all these tasks.

Using the CustomerOrders DataSet

Now comes the fun part — building the ASP.NET that will use all the items that were just created! The goal is to allow the end user to send in a request that contains just the CustomerID. In return, he will get back a complete DataSet containing not only the customer information, but also all the relevant order information. Listing 8-29 shows you the code to build all this functionality. You need only a single method in addition to the `Page_Load`: the `GetCustomerOrders()` method. The page should be laid out as is shown here in Figure 8-23.



Name: FK_Orders_Customers

Specify the keys that relate tables in your dataset.

Parent Table: Customers

Child Table: Orders

Columns:

Key Columns	Foreign Key Columns
CustomerID	CustomerID

Choose what to create

○ Both Relation and Foreign Key Constraint

○ Foreign Key Constraint Only

● Relation Only

Update Rule: None

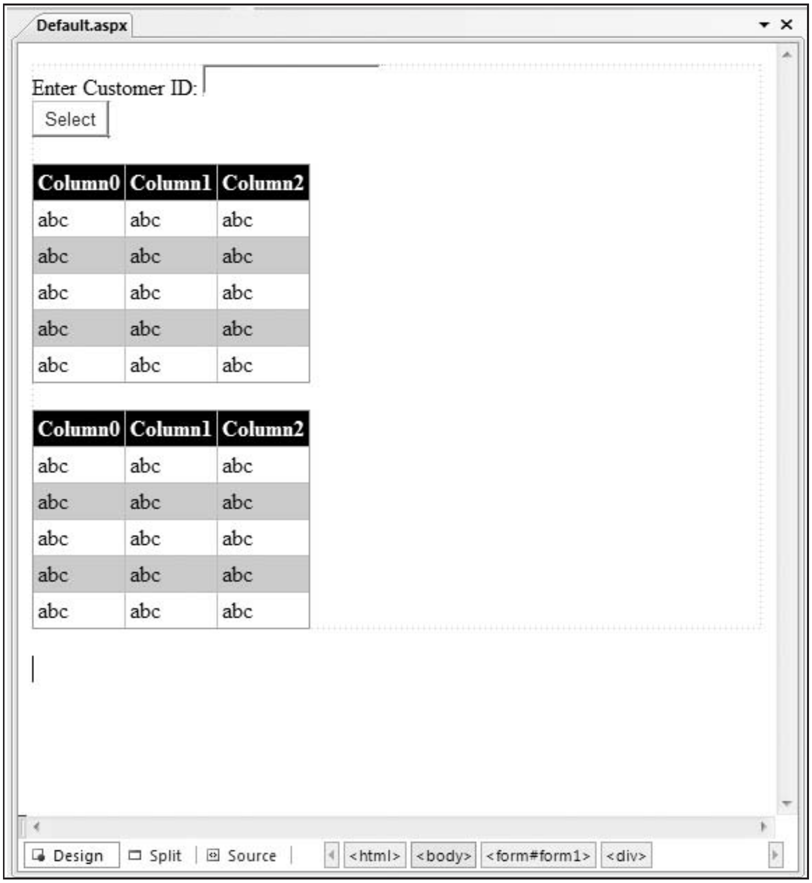
Delete Rule: None

Accept/Reject Rule: None

☐ Nested Relation

Figure 8-22

The page that you create should contain a single TextBox control, a Button control, and two GridView controls (GridView1 and GridView2). The code for the page is shown in Listing 8-29.



Default.aspx
Enter Customer ID:
Select
Column0
Column1
Column2
abc
Design
Split
Source
<html>
<body>
<form#form1>
<div>

Figure 8-23

Listing 8-29: The .aspx page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
    Inherits="_Default" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CustomerOrders</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        Enter Customer ID:
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <br />
```

Continued

```
            <asp:Button ID="Button1" runat="server" Text="Select" />
            <br />
            <br />
            <asp:GridView ID="GridView1" runat="server" BackColor="White"
                BorderColor="#999999" BorderStyle="Solid" BorderWidth="1px"
                CellPadding="3" ForeColor="Black" GridLines="Vertical">
                <FooterStyle BackColor="#CCCCCC" />
                <PagerStyle BackColor="#999999" ForeColor="Black"
                 HorizontalAlign="Center" />
                <SelectedRowStyle BackColor="#000099" Font-Bold="True"
                 ForeColor="White" />
                <HeaderStyle BackColor="Black" Font-Bold="True" ForeColor="White" />
                <AlternatingRowStyle BackColor="#CCCCCC" />
            </asp:GridView>
            <br />
            <asp:GridView ID="GridView2" runat="server" BackColor="White"
                BorderColor="#999999" BorderStyle="Solid" BorderWidth="1px"
                CellPadding="3" ForeColor="Black" GridLines="Vertical">
                <FooterStyle BackColor="#CCCCCC" />
                <PagerStyle BackColor="#999999" ForeColor="Black"
                 HorizontalAlign="Center" />
                <SelectedRowStyle BackColor="#000099" Font-Bold="True"
                 ForeColor="White" />
                <HeaderStyle BackColor="Black" Font-Bold="True" ForeColor="White" />
                <AlternatingRowStyle BackColor="#CCCCCC" />
            </asp:GridView>
        </div>
        </form>
</body>
</html>
```

The code-behind for the page is presented in Listing 8-30.

Listing 8-30: The code-behind for the CustomerOrders page

VB

```
Imports System

Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
     Handles Me.Load

        If Page.IsPostBack Then
            GetCustomerOrders(TextBox1.Text)
        End If
    End Sub

    Protected Sub GetCustomerOrders(ByVal custId As String)
        Dim myDataSet As New CustomerOrders
        Dim custDA As New CustomerOrdersTableAdapters.CustomersTableAdapter
        Dim ordersDA As New CustomerOrdersTableAdapters.OrdersTableAdapter
        custDA.Fill(myDataSet.Customers, custId)
        ordersDA.Fill(myDataSet.Orders, custId)

        myDataSet.Customers(0).Phone = "NOT AVAILABLE"
        myDataSet.Customers(0).Fax = "NOT AVAILABLE"

        GridView1.DataSource = myDataSet.Tables("Customers")
        GridView1.DataBind()

        GridView2.DataSource = myDataSet.Tables("Orders")
        GridView2.DataBind()
    End Sub
End Class
```

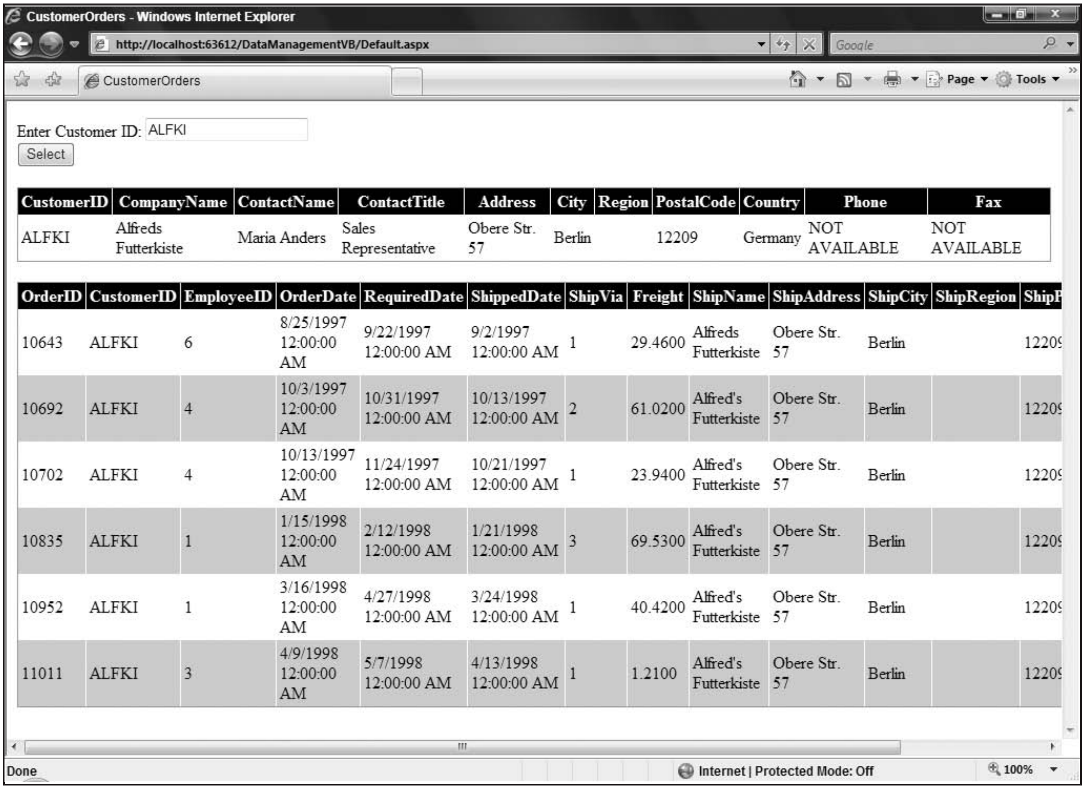
C#

```
using System;
using CustomerOrdersTableAdapters;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (Page.IsPostBack)
        {
            GetCustomerOrders(TextBox1.Text);
        }
    }

    protected void GetCustomerOrders(string custId)
    {
        CustomerOrders myDataSet = new CustomerOrders();
        CustomersTableAdapter custDA = new CustomersTableAdapter();
        OrdersTableAdapter ordersDA = new OrdersTableAdapter();

        custDA.Fill(myDataSet.Customers, custId);
        ordersDA.Fill(myDataSet.Orders, custId);

        myDataSet.Customers[0].Phone = "NOT AVAILABLE";
        myDataSet.Customers[0].Fax = "NOT AVAILABLE";

        GridView1.DataSource = myDataSet.Tables["Customers"];
        GridView1.DataBind();

        GridView2.DataSource = myDataSet.Tables["Orders"];
        GridView2.DataBind();
    }
}
```

Now there is not much code here. One of the first things done in the method is to create an instance of the typed DataSet. In the next two lines of code, the `custDA` and the `ordersDA` objects are used. In this case, the only accepted parameter, `custId`, is being set for both the DataAdapters. After this parameter is passed to the TableAdapter, this TableAdapter queries the database based upon the `select` query that you programmed into it earlier using the TableAdapter wizard.

After the query, the TableAdapter is instructed to fill the instance of the DataSet. Before the DataSet is returned to the consumer, you can change how the result is output to the client. If you are passing customer information, you may want to exclude some of the information. Because the DataSet is a typed DataSet, you have programmatic access to the tables. In this example, the code specifies that in the DataSet, in the Customers table, in the first row (remember it is zero-based), make the value of the Phone and Fax fields equal to `NOT AVAILABLE`.

By compiling and running the ASP.NET page, you are able to test it from the test page using the CustomerID of ALFKI (the first record of the Customers table in the Northwind database). The results are returned to you in the browser (see Figure 8-24).


CustomerOrders - Windows Internet Explorer
http://localhost:63612/DataManagementVB/Default.aspx
Google
CustomerOrders
Page
Tools
Select
Done
Internet | Protected Mode: Off
100%

Enter Customer ID: ALFKI

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	NOT AVAILABLE	NOT AVAILABLE

OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia	Freight	ShipName	ShipAddress	ShipCity	ShipRegion	ShipP
10643	ALFKI	6	8/25/1997 12:00:00 AM	9/22/1997 12:00:00 AM	9/2/1997 12:00:00 AM	1	29.4600	Alfreds Futterkiste	Obere Str. 57	Berlin		1220
10692	ALFKI	4	10/3/1997 12:00:00 AM	10/31/1997 12:00:00 AM	10/13/1997 12:00:00 AM	2	61.0200	Alfred's Futterkiste	Obere Str. 57	Berlin		1220
10702	ALFKI	4	10/13/1997 12:00:00 AM	11/24/1997 12:00:00 AM	10/21/1997 12:00:00 AM	1	23.9400	Alfred's Futterkiste	Obere Str. 57	Berlin		1220
10835	ALFKI	1	1/15/1998 12:00:00 AM	2/12/1998 12:00:00 AM	1/21/1998 12:00:00 AM	3	69.5300	Alfred's Futterkiste	Obere Str. 57	Berlin		1220
10952	ALFKI	1	3/16/1998 12:00:00 AM	4/27/1998 12:00:00 AM	3/24/1998 12:00:00 AM	1	40.4200	Alfred's Futterkiste	Obere Str. 57	Berlin		1220
11011	ALFKI	3	4/9/1998 12:00:00 AM	5/7/1998 12:00:00 AM	4/13/1998 12:00:00 AM	1	1.2100	Alfred's Futterkiste	Obere Str. 57	Berlin		1220

Figure 8-24

Asynchronous Command Execution

When you process data using ADO or previous versions of ADO.NET, each command is executed sequentially. The code waits for each command to complete before the next one is processed. When you use a single database, the sequential processing enables you to reuse the same connection object for all commands. However, with the introduction of MARS, you can now use a single connection for multiple, concurrent database access. Since the introduction of ADO.NET 2.0, ADO.NET has enabled users to process database commands asynchronously. This enables you to not only use the same connection, but also to use it in a parallel manner. The real advantage of asynchronous processing becomes apparent when you are accessing multiple data sources — especially when the data access queries across these databases aren't dependent on each other. You can now open a connection to the database in an asynchronous manner. When you are working with multiple databases, you can now open connections to them in a parallel fashion as well.

To make this work, be sure to add `Asynchronous Processing = true;` *to your connection string.*

Asynchronous Methods of the SqlCommand Class

The `SqlCommand` class provides a few additional methods that facilitate executing commands asynchronously. These new methods are summarized in the following table.

Method	Description
`BeginExecute-NonQuery()`	This method expects a query that doesn't return any results and starts it asynchronously. The return value is a reference to an object of the `SqlAsyncResult` class that implements the `IAsyncResult` interface. The returned object can be used to monitor the process as it runs and when it is completed.
`BeginExecuteNonQuery (callback, stateObject )`	This overloaded method also starts the process asynchronously, and it expects to receive an object of the `AsynchCallback` instance. The callback method is called after the process is finished running so that you can proceed with other tasks. The second parameter receives any custom-defined object. This object is passed to the callback automatically. It provides an excellent mechanism for passing parameters to the callback method. The callback method can retrieve the custom-defined state object by using the `AsyncState` property of the `IAsyncResult` interface.
`EndExecuteNonQuery (asyncResult)`	This method is used to access the results from the `BeginExecuteNonQuery` method. When calling this method, you are required to pass the same `SqlAsyncResult` object that you received when you called the `BeginExecuteNonQuery` method. This method returns an integer value containing the number of rows affected.
`BeginExecuteReader`	This method expects a query that returns a result set and starts it asynchronously. The return value is a reference to an object of `SqlAsyncResult` class that implements `IAsyncResult` interface. The returned object can be used to monitor the process as it runs and as it is completed.
`BeginExecuteReader (commandBehavior)`	This overloaded method works the same way as the one described previously. It also takes a parameter containing a command behavior enumeration just like the synchronous `ExecuteReader` method.
`BeginExecuteReader (callback, stateObject)`	This overloaded method starts the asynchronous process and it expects to receive an object of `AsyncCallback` instance. The callback method is called after the process finishes running so that you can proceed with other tasks. The second parameter receives any custom-defined object. This object is passed to the callback automatically. It provides an excellent mechanism for passing parameters to the callback method. The callback method can retrieve the custom-defined state object by using the `AsyncState` property of the `IAsyncResult` interface.

Method	Description
`BeginExecuteReader (callback, stateObject, commandBehavior)`	This overloaded method takes an instance of the `AsyncCallback` class and uses it to fire a callback method when the process has finished running. The second parameter receives a custom object to be passed to the callback method, and the third parameter uses the command behavior enumeration in the same way as the synchronous `ExecuteReader` method.
`EndExecuteReader`	This method is used to access the results from the `BeginExecuteReader` method. When calling this method, you are required to pass the same `SqlAsyncResult` object that you receive when you called the `BeginExecuteReader` method. This method returns a `SqlDataReader` object containing the result of the SQL query.
`BeginExecuteXmlReader`	This method expects a query that returns the result set as XML. The return value is a reference to an object of `SqlAsyncResult` class that implements `IAsyncResult` interface. The returned object can be used to monitor the process as it runs and as it is completed.
`BeginExecuteXmlReader (callback, stateObject)`	This overloaded method starts the asynchronous process, and it expects to receive an object of `AsyncCallback` instance. The callback method is called after the process has finished running so that you can proceed with other tasks. The second parameter receives any custom-defined object. This object is passed to the callback automatically. It provides an excellent mechanism for passing parameters to the callback method. The callback method can retrieve the custom-defined state object by using the `AsyncState` property of the `IAsyncResult` interface.
`EndExecuteXmlReader`	This method is used to access the results from the `BeginExecuteXmlReader` method. When calling this method, you are required to pass the same `SqlAsyncResult` object that you received when you called the `BeginExecuteXmlReader` method. This method returns an XML Reader object containing the result of the SQL query.

IAsyncResult Interface

All the asynchronous methods for the `SqlCommand` class return a reference to an object that exposes the `IAsyncResult` interface. The properties of this interface are shown in the following table.

Property	Description
`AsyncState`	This read-only property returns an object that describes the state of the process.
`AsyncWaitHandle`	This read-only property returns an instance of `WaitHandle` that can be used to set the time out, test whether the process has completed, and force the code to wait for completion.
`CompletedSynchronously`	This read-only property returns a Boolean value that indicates whether the process was executed synchronously.
`IsCompleted`	This read-only property returns a Boolean value indicating whether the process has completed.

AsyncCallback

Some of the asynchronous methods of the `SqlCommand` class receive an instance of the `AsyncCallback` class. This class is not specific to ADO.NET and is used by many objects in the .NET Framework. It is used to specify those methods that you want to execute after the asynchronous process has finished running. This class uses its constructor to receive the address of the method that you want to use for callback purposes.

WaitHandle Class

This class is an abstract class used for multiple purposes such as causing the execution to wait for any or all asynchronous processes to finish. To process more than one database command asynchronously, you can simply create an array containing wait handles for each asynchronous process. Using the static methods of the `WaitHandle` class, you can cause the execution to wait for either any or all wait handles in the array to finish processing.

The `WaitHandle` class exposes a few methods, as shown in the following table.

Method	Description
`WaitOne`	This method waits for a single asynchronous process to complete or time out. It returns a Boolean value containing `True` if the process completed successfully and `False` if it timed out.
`WaitOne (milliseconds, exitContext)`	This overloaded method receives an integer value as the first parameter. This value represents the time out in milliseconds. The second parameter receives a Boolean value specifying whether the method requires asynchronous context and should be set to `False` for asynchronous processing.
`WaitOne (timeSpan, exitContext)`	This overloaded method receives a `TimeSpan` object to represent the time-out value. The second parameter receives a Boolean value specifying whether the method requires asynchronous context and should be set to `False` for Asynchronous processing.
`WaitAny (waitHandles)`	This is a static method used if you are managing more than one `WaitHandle` in the form of an array. Using this method causes the execution to wait for any of the asynchronous processes that have been started and whose wait handles are in the array being passed to it. The `WaitAny` method must be called repeatedly — once for each `WaitHandle` you want to process.
`WaitAny (waitHandles, milliseconds, exitContext)`	This overloaded method receives the time-out value in the form of milliseconds and a Boolean value specifying whether the method requires asynchronous context. It should be set to `False` for asynchronous processing.
`WaitAny (waitHandles, timeSpan, exitContext`	This overloaded method receives the time-out value in the form of a `TimeSpan` object. The second parameter receives a Boolean value specifying whether the method requires asynchronous context. It should be set to `False` for asynchronous processing.
`WaitAll (waitHandles)`	This is a static method and is used to wait for all asynchronous processes to finish running.

Method	Description
`WaitAll (waitHandles, milliseconds, exitContext)`	This overloaded method receives the time-out value in the form of milliseconds and a Boolean value specifying whether the method requires asynchronous context. It should be set to `False` for asynchronous processing.
`WaitAll (waitHandles, timeSpan, exitContext)`	This overloaded method receives the time-out value in the form of `TimeSpan` object. The second parameter receives a Boolean value specifying whether the method requires asynchronous context. It should be set to `False` for asynchronous processing.
`Close ( )`	This method releases all wait handles and reclaims their resources.

Now that you understand asynchronous methods added to the `SqlCommand` and how to properly interact with them, you can write some code to see the asynchronous processing in action.

Approaches of Asynchronous Processing in ADO.NET

You can process asynchronous commands in three distinct ways. One approach is to start the asynchronous process and start polling the `IAsyncResult` object to see when the process has finished. The second approach is to provide a callback method while starting the asynchronous process. This approach enables you to perform other tasks in parallel. When the asynchronous process finishes, it fires the callback method that cleans up after the process and notifies other parts of the program that the asynchronous process has finished. The third and most elegant method is to associate a wait handle with the asynchronous process. Using this approach, you can start all the asynchronous processing you want and then wait for all or any of them to finish so that you can process them accordingly.

The Poll Approach

The code shown in Listing 8-31 creates an inline SQL statement to retrieve the top five records from the Orders table from the Northwind database. It starts the asynchronous process by calling the `BeginExecuteReader`. After the asynchronous process has started, it uses a `while` loop to wait for the process to finish. While waiting, the main thread sleeps for 10 milliseconds after checking the status of the asynchronous process. After the process has finished, it retrieves the result using the `EndExecuteReader` method.

Listing 8-31: The Poll approach to working with asynchronous commands

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim DBCon As SqlConnection
        Dim Command As SqlCommand = New SqlCommand()
        Dim OrdersReader As SqlDataReader
```

Continued

```
        Dim ASyncResult As IAsyncResult

        DBCon = New SqlConnection()
        DBCon.ConnectionString = _
           ConfigurationManager.ConnectionStrings("DSN_NorthWind").ConnectionString

        Command.CommandText = _
                "SELECT TOP 5 Customers.CompanyName, Customers.ContactName, " & _
                "Orders.OrderID, Orders.OrderDate, " & _
                "Orders.RequiredDate, Orders.ShippedDate " & _
                "FROM Orders, Customers " & _
                "WHERE Orders.CustomerID = Customers.CustomerID " & _
                "ORDER BY Customers.CompanyName, Customers.ContactName"
        Command.CommandType = CommandType.Text
        Command.Connection = DBCon

        DBCon.Open()

        ' Starting the asynchronous processing
        ASyncResult = Command.BeginExecuteReader()

        ' This loop with keep the main thread waiting until the
        ' asynchronous process is finished
        While Not ASyncResult.IsCompleted
            ' Sleeping current thread for 10 milliseconds
            System.Threading.Thread.Sleep(10)
        End While

        ' Retrieving result from the asynchronous process
        OrdersReader = Command.EndExecuteReader(ASyncResult)

        ' Displaying result on the screen
        gvOrders.DataSource = OrdersReader
        gvOrders.DataBind()

        ' Closing connection
        DBCon.Close()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>The Poll Approach</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:GridView ID="gvOrders" runat="server"
     AutoGenerateColumns="False" Width="100%">
       <Columns>
          <asp:BoundField HeaderText="Company Name"
           DataField="CompanyName"></asp:BoundField>
          <asp:BoundField HeaderText="Contact Name"
           DataField="ContactName"></asp:BoundField>
```

Continued

```
            <asp:BoundField HeaderText="Order Date"
             DataField="orderdate" DataFormatString="{0:d}"></asp:BoundField>
            <asp:BoundField HeaderText="Required Date" DataField="requireddate"
             DataFormatString="{0:d}"></asp:BoundField>
            <asp:BoundField HeaderText="Shipped Date" DataField="shippeddate"
             DataFormatString="{0:d}"></asp:BoundField>
         </Columns>
      </asp:GridView>
      </div>
      </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        SqlConnection DBCon;
        SqlCommand Command = new SqlCommand();
        SqlDataReader OrdersReader;
        IAsyncResult ASyncResult;

        DBCon = new SqlConnection();
        DBCon.ConnectionString =
           ConfigurationManager.ConnectionStrings["DSN_NorthWind"].ConnectionString;

        Command.CommandText =
                "SELECT TOP 5 Customers.CompanyName, Customers.ContactName, " +
                "Orders.OrderID, Orders.OrderDate, " +
                "Orders.RequiredDate, Orders.ShippedDate " +
                "FROM Orders, Customers " +
                "WHERE Orders.CustomerID = Customers.CustomerID " +
                "ORDER BY Customers.CompanyName, Customers.ContactName";

        Command.CommandType = CommandType.Text;
        Command.Connection = DBCon;

        DBCon.Open();

        // Starting the asynchronous processing
        ASyncResult = Command.BeginExecuteReader();

        // This loop with keep the main thread waiting until the
        // asynchronous process is finished
        while (!ASyncResult.IsCompleted)
        {
            // Sleeping current thread for 10 milliseconds
            System.Threading.Thread.Sleep(10);
        }
```

Continued

```
            // Retrieving result from the asynchronous process
            OrdersReader = Command.EndExecuteReader(ASyncResult);

            // Displaying result on the screen
            gvOrders.DataSource = OrdersReader;
            gvOrders.DataBind();

            // Closing connection
            DBCon.Close();
        }
    </script>
```

If you set a break point at the `while` loop, you will be able to see that the code execution continues after calling the `BeginExecuteReader` method. The code then continues to loop until the asynchronous execution has finished.

The Wait Approach

The most elegant of the three approaches is neither the poll approach nor the callback approach. The approach that provides the highest level of flexibility, efficiency, and (admittedly) a bit more complexity is the wait approach. Using this approach, you can write code that starts multiple asynchronous processes and waits for any or all the processes to finish running. This approach allows you to wait for only those processes that are dependent on each other and to proceed with the ones that don't. This approach, by its design, requires you to think about asynchronous processes in great detail. You must pick a good candidate for running in parallel and, most importantly, determine how different processes depend on each other. The complexity of this approach requires you to understand its details and design the code accordingly. The end result is, typically, a very elegant code design that makes the best use of synchronous and asynchronous processing models.

The code shown in Listing 8-32 uses the `WaitOne` method of the `WaitHandle` class. This method causes the program execution to wait until the asynchronous process has finished running.

Listing 8-32: The wait approach to handling a single asynchronous process

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim DBCon As SqlConnection
        Dim Command As SqlCommand = New SqlCommand()
        Dim OrdersReader As SqlDataReader
        Dim ASyncResult As IAsyncResult
        Dim WHandle As Threading.WaitHandle

        DBCon = New SqlConnection()
        DBCon.ConnectionString = _
           ConfigurationManager.ConnectionStrings("DSN_NorthWind").ConnectionString
```

Continued

```
        Command.CommandText = _
                "SELECT TOP 5 Customers.CompanyName, Customers.ContactName, " & _
                "Orders.OrderID, Orders.OrderDate, " & _
                "Orders.RequiredDate, Orders.ShippedDate " & _
                "FROM Orders, Customers " & _
                "WHERE Orders.CustomerID = Customers.CustomerID " & _
                "ORDER BY Customers.CompanyName, Customers.ContactName"

        Command.CommandType = CommandType.Text
        Command.Connection = DBCon

        DBCon.Open()

        ' Starting the asynchronous processing
        ASyncResult = Command.BeginExecuteReader()

        WHandle = ASyncResult.AsyncWaitHandle

        If WHandle.WaitOne = True Then
            ' Retrieving result from the asynchronous process
            OrdersReader = Command.EndExecuteReader(ASyncResult)

            ' Displaying result on the screen
            gvOrders.DataSource = OrdersReader
            gvOrders.DataBind()

            ' Closing connection
            DBCon.Close()
        Else
            ' Asynchronous process has timed out. Handle this
            ' situation here.
        End If
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>The Wait Approach</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:GridView ID="gvOrders" runat="server"
     AutoGenerateColumns="False" Width="100%">
       <Columns>
          <asp:BoundField HeaderText="Company Name"
           DataField="CompanyName"></asp:BoundField>
          <asp:BoundField HeaderText="Contact Name"
           DataField="ContactName"></asp:BoundField>
          <asp:BoundField HeaderText="Order Date"
           DataField="orderdate" DataFormatString="{0:d}"></asp:BoundField>
          <asp:BoundField HeaderText="Required Date" DataField="requireddate"
           DataFormatString="{0:d}"></asp:BoundField>
```

Continued

```
          <asp:BoundField HeaderText="Shipped Date" DataField="shippeddate"
           DataFormatString="{0:d}"></asp:BoundField>
       </Columns>
    </asp:GridView>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        SqlConnection DBCon;
        SqlCommand Command = new SqlCommand();
        SqlDataReader OrdersReader;
        IAsyncResult ASyncResult;
        System.Threading.WaitHandle WHandle;

        DBCon = new SqlConnection();
        DBCon.ConnectionString =
           ConfigurationManager.ConnectionStrings["DSN_NorthWind"].ConnectionString;

        Command.CommandText =
                "SELECT TOP 5 Customers.CompanyName, Customers.ContactName, " +
                "Orders.OrderID, Orders.OrderDate, " +
                "Orders.RequiredDate, Orders.ShippedDate " +
                "FROM Orders, Customers " +
                "WHERE Orders.CustomerID = Customers.CustomerID " +
                "ORDER BY Customers.CompanyName, Customers.ContactName";

        Command.CommandType = CommandType.Text;
        Command.Connection = DBCon;

        DBCon.Open();

        // Starting the asynchronous processing
        ASyncResult = Command.BeginExecuteReader();

        WHandle = ASyncResult.AsyncWaitHandle;

        if (WHandle.WaitOne() == true)
        {
            // Retrieving result from the asynchronous process
            OrdersReader = Command.EndExecuteReader(ASyncResult);

            // Displaying result on the screen
            gvOrders.DataSource = OrdersReader;
            gvOrders.DataBind();
```

Continued

```
            // Closing connection
            DBCon.Close();
        }
        else
        {
            // Asynchronous process has timed out. Handle this
            // situation here.
        }
    }
</script>
```

If you set a break point and step through this code, you will notice that the program execution stops at the `WHandle.WaitOne` method call. The program automatically resumes when the asynchronous commands finishes its execution.

Using Multiple Wait Handles

The real power of the wait approach doesn't become apparent until you start multiple asynchronous processes. The code shown in Listing 8-33 starts two asynchronous processes. One process queries a database to get information about a specific customer and runs another query to retrieve all orders submitted by that the same customer. The code example shown in this listing creates two separate `Command` objects, Data Reader objects, and wait handles. However, it uses the same connection object for both queries to demonstrate how well Multiple Active Result Set (MARS) supports work in conjunction with the asynchronous processing.

Listing 8-33: Use of multiple wait handles in conjunction with MARS

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim DBCon As SqlConnection
        Dim OrdersCommand As SqlCommand = New SqlCommand()
        Dim CustCommand As SqlCommand = New SqlCommand()
        Dim OrdersReader As SqlDataReader
        Dim CustReader As SqlDataReader
        Dim OrdersASyncResult As IAsyncResult
        Dim CustAsyncResult As IAsyncResult

        Dim WHandles(1) As System.Threading.WaitHandle
        Dim OrdersWHandle As System.Threading.WaitHandle
        Dim CustWHandle As System.Threading.WaitHandle

        DBCon = New SqlConnection()
        DBCon.ConnectionString = _
           ConfigurationManager.ConnectionStrings("DSN_NorthWind").ConnectionString
```

Continued

```
        CustCommand.CommandText = _
           "SELECT * FROM Customers WHERE CompanyName = 'Alfreds Futterkiste'"

        CustCommand.CommandType = CommandType.Text
        CustCommand.Connection = DBCon

        ' Selecting all orders for a specific customer
        OrdersCommand.CommandText = _
                "SELECT Customers.CompanyName, Customers.ContactName, " & _
                "Orders.OrderID, Orders.OrderDate, " & _
                "Orders.RequiredDate, Orders.ShippedDate " & _
                "FROM Orders, Customers " & _
                "WHERE Orders.CustomerID = Customers.CustomerID " & _
                "AND Customers.CompanyName = 'Alfreds Futterkiste' " & _
                "ORDER BY Customers.CompanyName, Customers.ContactName"

        OrdersCommand.CommandType = CommandType.Text
        OrdersCommand.Connection = DBCon

        DBCon.Open()

        ' Retrieving customer information asynchronously
        CustAsyncResult = CustCommand.BeginExecuteReader()

        ' Retrieving orders list asynchronously
        OrdersASyncResult = OrdersCommand.BeginExecuteReader()

        CustWHandle = CustAsyncResult.AsyncWaitHandle
        OrdersWHandle = OrdersASyncResult.AsyncWaitHandle

        ' Filling Wait Handles array with the two wait handles we
        ' are going to use in this code
        WHandles(0) = CustWHandle
        WHandles(1) = OrdersWHandle

        System.Threading.WaitHandle.WaitAll(WHandles)

        CustReader = CustCommand.EndExecuteReader(CustAsyncResult)

        OrdersReader = OrdersCommand.EndExecuteReader(OrdersASyncResult)

        gvCustomers.DataSource = CustReader
        gvCustomers.DataBind()

        gvOrders.DataSource = OrdersReader
        gvOrders.DataBind()

        DBCon.Close()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
```

Continued

```
    <title>Wait All Approach</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:GridView ID="gvCustomers" Width="100%" runat="server"></asp:GridView>
    <br /><br />
    <asp:GridView ID="gvOrders" Width="100%" AutoGenerateColumns="False"
     runat="server">
       <Columns>
          <asp:BoundField HeaderText="Company Name"
           DataField="CompanyName"></asp:BoundField>
          <asp:BoundField HeaderText="Contact Name"
           DataField="ContactName"></asp:BoundField>
          <asp:BoundField HeaderText="Order Date" DataField="orderdate"
           DataFormatString="{0:d}"></asp:BoundField>
          <asp:BoundField HeaderText="Required Date" DataField="requireddate"
           DataFormatString="{0:d}"></asp:BoundField>
          <asp:BoundField HeaderText="Shipped Date" DataField="shippeddate"
           DataFormatString="{0:d}"></asp:BoundField>
       </Columns>
    </asp:GridView>
    </div>
    </form>
</body>
</html>
```

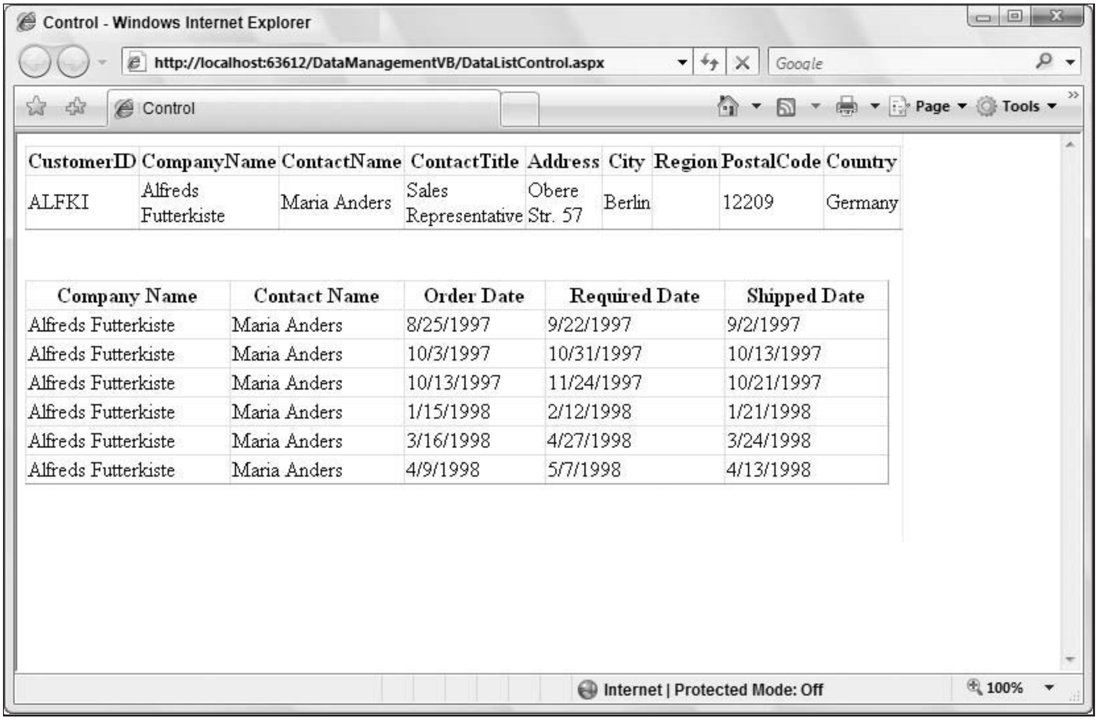
C#

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        SqlConnection DBCon;
        SqlCommand OrdersCommand = new SqlCommand();
        SqlCommand CustCommand = new SqlCommand();
        SqlDataReader OrdersReader;
        SqlDataReader CustReader;
        IAsyncResult OrdersASyncResult;
        IAsyncResult CustAsyncResult;

        System.Threading.WaitHandle[] WHandles = new
           System.Threading.WaitHandle[1];
        System.Threading.WaitHandle OrdersWHandle;
        System.Threading.WaitHandle CustWHandle;

        DBCon = new SqlConnection();
        DBCon.ConnectionString =
           ConfigurationManager.ConnectionStrings["DSN_NorthWind"].ConnectionString;
```

Continued

```
            CustCommand.CommandText =
               "SELECT * FROM Customers WHERE CompanyName = 'Alfreds Futterkiste'";

            CustCommand.CommandType = CommandType.Text;
            CustCommand.Connection = DBCon;

            // Selecting all orders for a specific customer
            OrdersCommand.CommandText =
                    "SELECT Customers.CompanyName, Customers.ContactName, " +
                    "Orders.OrderID, Orders.OrderDate, " +
                    "Orders.RequiredDate, Orders.ShippedDate " +
                    "FROM Orders, Customers " +
                    "WHERE Orders.CustomerID = Customers.CustomerID " +
                    "AND Customers.CompanyName = 'Alfreds Futterkiste' " +
                    "ORDER BY Customers.CompanyName, Customers.ContactName";

            OrdersCommand.CommandType = CommandType.Text;
            OrdersCommand.Connection = DBCon;

            DBCon.Open();

            // Retrieving customer information asynchronously
            CustAsyncResult = CustCommand.BeginExecuteReader();

            // Retrieving orders list asynchronously
            OrdersASyncResult = OrdersCommand.BeginExecuteReader();

            CustWHandle = CustAsyncResult.AsyncWaitHandle;
            OrdersWHandle = OrdersASyncResult.AsyncWaitHandle;

            // Filling Wait Handles array with the two wait handles we
            // are going to use in this code
            WHandles[0] = CustWHandle;
            WHandles[1] = OrdersWHandle;

            System.Threading.WaitHandle.WaitAll(WHandles);

            CustReader = CustCommand.EndExecuteReader(CustAsyncResult);

            OrdersReader = OrdersCommand.EndExecuteReader(OrdersASyncResult);

            gvCustomers.DataSource = CustReader;
            gvCustomers.DataBind();

            gvOrders.DataSource = OrdersReader;
            gvOrders.DataBind();

            DBCon.Close();
        }
    </script>
```

When you compile and execute the code shown in Listing 8-33, you see the result on the screen, as shown in Figure 8-25. This figure clearly shows two GridView controls that were used in the code example. The GridView control on the top shows the result of executing a query that retrieved all information related to a specific customer. The GridView control on the bottom shows the results of executing the second query that retrieved a list of all orders submitted by a specific customer.


Control - Windows Internet Explorer
http://localhost:63612/DataManagementVB/DataListControl.aspx
Google
Control
Page
Tools
Internet | Protected Mode: Off
100%

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany

Company Name	Contact Name	Order Date	Required Date	Shipped Date
Alfreds Futterkiste	Maria Anders	8/25/1997	9/22/1997	9/2/1997
Alfreds Futterkiste	Maria Anders	10/3/1997	10/31/1997	10/13/1997
Alfreds Futterkiste	Maria Anders	10/13/1997	11/24/1997	10/21/1997
Alfreds Futterkiste	Maria Anders	1/15/1998	2/12/1998	1/21/1998
Alfreds Futterkiste	Maria Anders	3/16/1998	4/27/1998	3/24/1998
Alfreds Futterkiste	Maria Anders	4/9/1998	5/7/1998	4/13/1998

Figure 8-25

The code shown in Listing 8-33 reveals some of the elegance of using the wait approach. However, it is still not the most efficient code you can write with ADO.NET 2.0. The code should allow for a wait until both asynchronous processes finish running before the data binds the result sets to the respective GridView controls.

You can change the code shown in Listing 8-33 just a little to gain even more efficiency. Replace the `WaitAll` method with the `WaitAny` method. The `WaitAny` method enables you to handle the results of each of the asynchronous processes as soon as each is completed without waiting for other processing to finish. To use the `WaitAny` method and still manage the execution of all asynchronous processes, you can also add a loop that enables you to make sure that all asynchronous processes are handled after they are completed.

The `WaitAny` method returns an Integer value that indicates an array index of the wait handle that has finished running. Using this return value, you can easily find the correct wait handle and process the result set retrieved from the query that was executed in that particular process, as shown in Listing 8-34.

Listing 8-34: Use of the WaitAny method to process multiple asynchronous processes

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim DBCon As SqlConnection
        Dim OrdersCommand As SqlCommand = New SqlCommand()
        Dim CustCommand As SqlCommand = New SqlCommand()
        Dim OrdersReader As SqlDataReader
        Dim CustReader As SqlDataReader
        Dim OrdersASyncResult As IAsyncResult
        Dim CustAsyncResult As IAsyncResult

        Dim WHIndex As Integer
        Dim WHandles(1) As Threading.WaitHandle
        Dim OrdersWHandle As Threading.WaitHandle
        Dim CustWHandle As Threading.WaitHandle

        DBCon = New SqlConnection()
        DBCon.ConnectionString = _
           ConfigurationManager.ConnectionStrings("DSN_NorthWind").ConnectionString

        CustCommand.CommandText = _
           "SELECT * FROM Customers WHERE CompanyName = 'Alfreds Futterkiste'"

        CustCommand.CommandType = CommandType.Text
        CustCommand.Connection = DBCon

        OrdersCommand.CommandText = _
                "SELECT Customers.CompanyName, Customers.ContactName, " & _
                "Orders.OrderID, Orders.OrderDate, " & _
                "Orders.RequiredDate, Orders.ShippedDate " & _
                "FROM Orders, Customers " & _
                "WHERE Orders.CustomerID = Customers.CustomerID " & _
                "AND Customers.CompanyName = 'Alfreds Futterkiste' " & _
                "ORDER BY Customers.CompanyName, Customers.ContactName"

        OrdersCommand.CommandType = CommandType.Text
        OrdersCommand.Connection = DBCon

        ' Opening the database connection
        DBCon.Open ()
```

Continued

```
        ' Retrieving customer information asynchronously
        CustAsyncResult = CustCommand.BeginExecuteReader()

        ' Retrieving orders list asynchronously
        OrdersASyncResult = OrdersCommand.BeginExecuteReader()

        CustWHandle = CustAsyncResult.AsyncWaitHandle
        OrdersWHandle = OrdersASyncResult.AsyncWaitHandle

        ' Filling Wait Handles array with the two wait handles we
        ' are going to use in this code
        WHandles(0) = CustWHandle
        WHandles(1) = OrdersWHandle

        ' Looping 2 times because there are 2 wait handles
        ' in the array
        For Index As Integer = 0 To 1
            ' We are only waiting for any of the two
            ' asynchronous process to finish running
            WHIndex = Threading.WaitHandle.WaitAny(WHandles)

            ' The return value from the WaitAny method is
            ' the array index of the Wait Handle that just
            ' finsihed running
            Select Case WHIndex
                Case 0
                    CustReader = CustCommand.EndExecuteReader(CustAsyncResult)

                    gvCustomers.DataSource = CustReader
                    gvCustomers.DataBind()
                Case 1
                    OrdersReader = _
                       OrdersCommand.EndExecuteReader(OrdersASyncResult)

                    gvOrders.DataSource = OrdersReader
                    gvOrders.DataBind()

            End Select
        Next

        ' Closing connection
        DBCon.Close()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>The Wait Any Approach</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:GridView ID="gvCustomers" Width="100%" runat="server"></asp:GridView>
    <br /><br />
```

Continued

```
    <asp:GridView ID="gvOrders" Width="100%" AutoGenerateColumns="False"
     runat="server">
       <Columns>
          <asp:BoundField HeaderText="Company Name"
           DataField="CompanyName"></asp:BoundField>
          <asp:BoundField HeaderText="Contact Name"
           DataField="ContactName"></asp:BoundField>
          <asp:BoundField HeaderText="Order Date" DataField="orderdate"
           DataFormatString="{0:d}"></asp:BoundField>
          <asp:BoundField HeaderText="Required Date" DataField="requireddate"
           DataFormatString="{0:d}"></asp:BoundField>
          <asp:BoundField HeaderText="Shipped Date" DataField="shippeddate"
           DataFormatString="{0:d}"></asp:BoundField>
       </Columns>
    </asp:GridView>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        SqlConnection DBCon;
        SqlCommand OrdersCommand = new SqlCommand();
        SqlCommand CustCommand = new SqlCommand();
        SqlDataReader OrdersReader;
        SqlDataReader CustReader;
        IAsyncResult OrdersASyncResult;
        IAsyncResult CustAsyncResult;

        int WHIndex;
        System.Threading.WaitHandle[] WHandles =
           new System.Threading.WaitHandle[1];
        System.Threading.WaitHandle OrdersWHandle;
        System.Threading.WaitHandle CustWHandle;

        DBCon = new SqlConnection();
        DBCon.ConnectionString =
           ConfigurationManager.ConnectionStrings["DSN_NorthWind"].ConnectionString;

        CustCommand.CommandText =
           "SELECT * FROM Customers WHERE CompanyName = 'Alfreds Futterkiste'";

        CustCommand.CommandType = CommandType.Text;
        CustCommand.Connection = DBCon;

        OrdersCommand.CommandText =
```

Continued

```
                "SELECT Customers.CompanyName, Customers.ContactName, " +
                "Orders.OrderID, Orders.OrderDate, " +
                "Orders.RequiredDate, Orders.ShippedDate " +
                "FROM Orders, Customers " +
                "WHERE Orders.CustomerID = Customers.CustomerID " +
                "AND Customers.CompanyName = 'Alfreds Futterkiste' " +
                "ORDER BY Customers.CompanyName, Customers.ContactName";

        OrdersCommand.CommandType = CommandType.Text;
        OrdersCommand.Connection = DBCon;

        // Opening the database connection
        DBCon.Open();

        // Retrieving customer information asynchronously
        CustAsyncResult = CustCommand.BeginExecuteReader();

        // Retrieving orders list asynchronously
        OrdersASyncResult = OrdersCommand.BeginExecuteReader();

        CustWHandle = CustAsyncResult.AsyncWaitHandle;
        OrdersWHandle = OrdersASyncResult.AsyncWaitHandle;

        // Filling Wait Handles array with the two wait handles we
        // are going to use in this code
        WHandles[0] = CustWHandle;
        WHandles[1] = OrdersWHandle;

        // Looping 2 times because there are 2 wait handles
        // in the array
        for (int Index = 0; Index < 2; Index++ )
        {
            // We are only waiting for any of the two
            // asynchronous process to finish running
            WHIndex = System.Threading.WaitHandle.WaitAny(WHandles);

            // The return value from the WaitAny method is
            // the array index of the Wait Handle that just
            // finsihed running
            switch (WHIndex)
            {
                case 0:
                    CustReader = CustCommand.EndExecuteReader(CustAsyncResult);

                    gvCustomers.DataSource = CustReader;
                    gvCustomers.DataBind();
                    break;
                case 1:
                    OrdersReader =
                       OrdersCommand.EndExecuteReader(OrdersASyncResult);

                    gvOrders.DataSource = OrdersReader;
                    gvOrders.DataBind();
                    break;
```

Continued

```
            }
        }
        // Closing connection
        DBCon.Close();
    }
</script>
```

Next, look at the callback approach. Using this approach, you assign a callback method to the asynchronous process and use it to display the result returned by executing the SQL query.

The Callback Approach

Listing 8-35 creates an inline SQL statement that retrieves the top five records from the database. It starts the asynchronous process by calling the `BeginExecuteReader` method and passing it the callback delegate. No further processing is needed, and the method ends after the asynchronous process has started. After the callback method is fired, it retrieves the result and displays it on the screen.

Listing 8-35: Asynchronous command processing using the callback approach

VB

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    Private Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim DBCon As SqlConnection
        Dim Command As SqlCommand = New SqlCommand()
        Dim ASyncResult As SqlAsyncResult

        DBCon = New SqlConnection()
        Command = New SqlCommand()
        DBCon.ConnectionString = _
          ConfigurationManager.ConnectionStrings("DSN_NorthWind").ConnectionString

        ' Selecting top 5 records from the Orders table
        Command.CommandText = _
                "SELECT TOP 5 Customers.CompanyName, Customers.ContactName, " & _
                "Orders.OrderID, Orders.OrderDate, " & _
                "Orders.RequiredDate, Orders.ShippedDate " & _
                "FROM Orders, Customers " & _
                "WHERE Orders.CustomerID = Customers.CustomerID " & _
                "ORDER BY Customers.CompanyName, Customers.ContactName"

        Command.CommandType = CommandType.Text
        Command.Connection = DBCon

        DBCon.Open()

        ' Starting the asynchronous processing
        AsyncResult = Command.BeginExecuteReader(New _
           AsyncCallback(AddressOf CBMethod), CommandBehavior.CloseConnection)
```

Continued

```
    End Sub

    Public Sub CBMethod(ByVal ar As SQLAsyncResult)
        Dim OrdersReader As SqlDataReader

        ' Retrieving result from the asynchronous process
        OrdersReader = ar.EndExecuteReader(ar)

        ' Displaying result on the screen
        gvOrders.DataSource = OrdersReader
        gvOrders.DataBind()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>The Call Back Approach</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:GridView ID="gvOrders" Width="100%" AutoGenerateColumns="False"
     runat="server">
       <Columns>
          <asp:BoundField HeaderText="Company Name"
           DataField="CompanyName"></asp:BoundField>
          <asp:BoundField HeaderText="Contact Name"
           DataField="ContactName"></asp:BoundField>
          <asp:BoundField HeaderText="Order Date" DataField="orderdate"
           DataFormatString="{0:d}"></asp:BoundField>
          <asp:BoundField HeaderText="Required Date" DataField="requireddate"
           DataFormatString="{0:d}"></asp:BoundField>
          <asp:BoundField HeaderText="Shipped Date" DataField="shippeddate"
           DataFormatString="{0:d}"></asp:BoundField>
       </Columns>
    </asp:GridView>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        SqlConnection DBCon;
```

Continued

```
            SqlCommand Command = new SqlCommand();
            SqlAsyncResult ASyncResult;
            DBCon = new SqlConnection();
            Command = new SqlCommand();
            DBCon.ConnectionString =
               ConfigurationManager.ConnectionStrings["DSN_NorthWind"].ConnectionString;

            // Selecting top 5 records from the Orders table
            Command.CommandText =
                    "SELECT TOP 5 Customers.CompanyName, Customers.ContactName, " +
                    "Orders.OrderID, Orders.OrderDate, " +
                    "Orders.RequiredDate, Orders.ShippedDate " +
                    "FROM Orders, Customers " +
                    "WHERE Orders.CustomerID = Customers.CustomerID " +
                    "ORDER BY Customers.CompanyName, Customers.ContactName";

            Command.CommandType = CommandType.Text;
            Command.Connection = DBCon;

            DBCon.Open();

            // Starting the asynchronous processing
            AsyncResult = Command.BeginExecuteReader(new AsyncCallback(CBMethod),
               CommandBehavior.CloseConnection);
        }

        public void CBMethod(SQLAsyncResult ar)
        {
            SqlDataReader OrdersReader;

            // Retrieving result from the asynchronous process
            OrdersReader = ar.EndExecuteReader(ar);

            // Displaying result on the screen
            gvOrders.DataSource = OrdersReader;
            gvOrders.DataBind();
        }
    </script>
```

The callback approach enables you to handle the result of a command execution at a different part of your code. This feature is useful in cases where the command execution takes longer than usual and you want to respond to the user without waiting for the command execution to finish.

Canceling Asynchronous Processing

The asynchronous process often takes longer than expected. To alleviate this problem, you can provide an option to the user to cancel the process without waiting for the result. Canceling an asynchronous process is as easy as calling the `Cancel` method on the appropriate `Command` object. This method doesn't return any value. To roll back the work that was already completed by the `Command` object, you must provide a custom transaction to the `Command` object before executing the query. You can also handle the rollback or the commit process yourself.

Asynchronous Connections

Now that you understand how to execute multiple database queries asynchronously using the `Command` object, take a quick look at how you can open database connections asynchronously, as well. The principles of working with asynchronous connections are the same as when you work with asynchronous commands. You can still use any of the three approaches you learned previously.

In ADO.NET, the `SqlConnection` class exposes a couple of new properties needed when working asynchronously. These properties are shown in the following table.

Property	**Description**
`Asynchronous`	This read-only property returns a Boolean value indicating whether the connection has been opened asynchronously.
`State`	This property returns a value from `System.Data.ConnectionState` enumeration indicating the state of the connection. The possible values are as follows: ❑ `Broken` ❑ `Closed` ❑ `Connecting` ❑ `Executing` ❑ `Fetching` ❑ `Open`

Summary

In summary, ADO.NET is a powerful tool to incorporate within your ASP.NET applications. ADO.NET has a number of new technologies that provide you with data solutions that you could only dream of in the past.

Visual Studio also makes ADO.NET programming quick and easy when you use the wizards that are available. In this chapter, you saw a number of the wizards. You do not have to use these wizards in order to work with ADO.NET. On the contrary, you can use some of the wizards and create the rest of the code yourself, or you can use none of the wizards. In any case, you have complete and full access to everything that ADO.NET provides.

This chapter covered a range of advanced features of ADO.NET as well. These features are designed to give you the flexibility to handle database processing in a manner never before possible with either of the previous versions of ADO.NET or ADO.

This chapter also covered the features of Multiple Active Result Sets (MARS), which enables you to reuse a single open connection for multiple accesses to the database, even if the connection is currently processing a result set. This feature becomes even more powerful when it is used in conjunction with the asynchronous command processing.

9

Querying with LINQ

.NET 3.5 introduces a new technology called Language Integrated Query, or LINQ (pronounced ‘‘link’’). LINQ is designed to fill the gap that exists between traditional .NET languages, which offer strong typing and full object-oriented development, and query languages such as SQL, with syntax specifically designed for query operations. With the introduction of LINQ into .NET, query becomes a first class concept in .NET, whether object, XML, or data queries.

LINQ includes three basic types of queries, LINQ to Objects, LINQ to XML (or XLINQ), and LINQ to SQL (or DLINQ). Each type of query offers specific capabilities and is designed to query a specific source.

In this chapter, we look at all three flavors of LINQ, and how each enables you to simplify query operations. We also look at some new language features of the .NET CLR that you will use to create LINQ queries, as well as the tooling support added to Visual Studio to support using LINQ.

LINQ to Objects

The first and most basic flavor of LINQ is LINQ to Objects. LINQ to Objects enables you to perform complex query operations against any enumerable object (any object that implements the IEnumerable interface). While the notion of creating enumerable objects that can be queried or sorted is not new to .NET, doing this in versions prior to version 3.5 usually required a significant amount of code. Often that code would end up being so complex that it would be hard for other developers to read and understand, making it difficult to maintain.

Traditional Query Methods

In order to really understand how LINQ improves your ability to query collections, you really need to understand how this is done without it. To do this, let’s take a look at how you might create a simple query that includes a group and sort without using LINQ. Listing 9-1 shows a simple Movie class you can use as the basis of these examples.

Listing 9-1: A basic Movie class

VB

```
Imports Microsoft.VisualBasic

Public Class Movie
    Private _title As String
    Private _director As String
    Private _genre As Integer
    Private _runtime As Integer
    Private _releasedate As DateTime

    Public Property Title() As String
        Get
            Return _title
        End Get
        Set(ByVal value As String)
            _title = value
        End Set
    End Property

    Public Property Director() As String
        Get
            Return _director
        End Get
        Set(ByVal value As String)
            _director = value
        End Set
    End Property

    Public Property Genre() As Integer
        Get
            Return _genre
        End Get
        Set(ByVal value As Integer)
            _genre = value
        End Set
    End Property

    Public Property Runtime() As Integer
        Get
            Return _runtime
        End Get
        Set(ByVal value As Integer)
            _runtime = value
        End Set
    End Property

    Public Property ReleaseDate() As DateTime
        Get
            Return _releasedate
        End Get
        Set(ByVal value As DateTime)
```

```
            _releasedate = value
        End Set
    End Property
End Class
```

C#

```
using System;

public class Movie
{
    public string Title { get; set; }
    public string Director { get; set; }
    public int Genre { get; set; }
    public int RunTime { get; set; }
    public DateTime ReleaseDate { get; set; }
}
```

This is the basic class that is used throughout this section and the following LINQ to Object section.

Now that you have a basic class to work with, let's look at how you would normally use the class. Listing 9-2 demonstrates how to create a simple generic List of the Movie objects in an ASP.NET page, and then binding that list to a GridView control. The GridView displays the values of all public properties exposed by the Movie class.

Listing 9-2: Generating a list of Movie objects and binding to a GridView

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Collections.Generic" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim movies = GetMovies()

        Me.GridView1.DataSource = movies
        Me.GridView1.DataBind()
    End Sub

    Public Function GetMovies() As List(Of Movie)
      Dim movies As Movie() = { _
          New Movie With {.Title = "Shrek", .Director = "Andrew Adamson", _
                .Genre = 0, .ReleaseDate = DateTime.Parse("5/16/2001"),
.Runtime = 89}, _
          New Movie With {.Title = "Fletch", .Director = "Michael Ritchie", _
               .Genre = 0, .ReleaseDate = DateTime.Parse("5/31/1985"),
.Runtime = 96}, _
          New Movie With {.Title = "Casablanca", .Director = "Michael Curtiz", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("1/1/1942"),
.Runtime = 102}, _
          New Movie With {.Title = "Batman", .Director = "Tim Burton", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("6/23/1989"),
.Runtime = 126}, _
```

Continued

```
          New Movie With {.Title = "Dances with Wolves",
.Director = "Kevin Costner", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("11/21/1990"),
.Runtime = 180}, _
          New Movie With {.Title = "Dirty Dancing", .Director = "Emile Ardolino", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("8/21/1987"),
.Runtime = 100}, _
          New Movie With {.Title = "The Parent Trap", .Director = "Nancy Meyers", _
                .Genre = 0, .ReleaseDate = DateTime.Parse("7/29/1998"),
.Runtime = 127}, _
          New Movie With {.Title = "Ransom", .Director = "Ron Howard", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("11/8/1996"),
.Runtime = 121}, _
         New Movie With {.Title = "Ocean's Eleven", .Director = "Steven Soderbergh", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("12/7/2001"),
.Runtime = 116}, _
          New Movie With {.Title = "Steel Magnolias", .Director = "Herbert Ross", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("11/15/1989"),
.Runtime = 117}, _
          New Movie With {.Title = "Mystic Pizza", .Director = "Donald Petrie", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("10/21/1988"),
.Runtime = 104}, _
          New Movie With {.Title = "Pretty Woman", .Director = "Garry Marshall", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("3/23/1990"),
.Runtime = 119}, _
          New Movie With {.Title = "Interview with the Vampire",
.Director = "Neil Jordan", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("11/11/1994"),
.Runtime = 123}, _
          New Movie With {.Title = "Top Gun", .Director = "Tony Scott", _
                .Genre = 2, .ReleaseDate = DateTime.Parse("5/16/1986"),
.Runtime = 110}, _
          New Movie With {.Title = "Mission Impossible",
.Director = "Brian De Palma", _
                .Genre = 2, .ReleaseDate = DateTime.Parse("5/22/1996"),
.Runtime = 110}, _
          New Movie With {.Title = "The Godfather",
.Director = "Francis Ford Coppola", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("3/24/1972"),
.Runtime = 175}, _
          New Movie With {.Title = "Carlito's Way", .Director = "Brian De Palma", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("11/10/1993"),
.Runtime = 144}, _
          New Movie With {.Title = "Robin Hood: Prince of Thieves", _
                .Director = "Kevin Reynolds", .Genre = 1, _
                .ReleaseDate = DateTime.Parse("6/14/1991"), .Runtime = 143}, _
          New Movie With {.Title = "The Haunted", .Director = "Robert Mandel", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("5/6/1991"),
.Runtime = 100}, _
          New Movie With {.Title = "Old School", .Director = "Todd Phillips", _
                .Genre = 0, .ReleaseDate = DateTime.Parse("2/21/2003"),
.Runtime = 91}, _
          New Movie With {.Title = "Anchorman: The Legend of Ron Burgundy", _
```

Continued

```
               .Director = "Adam McKay", .Genre = 0, _
               .ReleaseDate = DateTime.Parse("7/9/2004"),
.Runtime = 94}, _
          New Movie With {.Title = "Bruce Almighty", .Director = "Tom Shadyac", _
                .Genre = 0, .ReleaseDate = DateTime.Parse("5/23/2003"),
.Runtime = 101}, _
          New Movie With {.Title = "Ace Ventura: Pet Detective", _
                .Director = "Tom Shadyac", .Genre = 0, _
                .ReleaseDate = DateTime.Parse("2/4/1994"),
.Runtime = 86}, _
          New Movie With {.Title = "Goonies", .Director = "Richard Donner", _
                .Genre = 0, .ReleaseDate = DateTime.Parse("6/7/1985"),
.Runtime = 114}, _
          New Movie With {.Title = "Sixteen Candles", .Director = "John Hughes", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("5/4/1984"),
.Runtime = 93}, _
         New Movie With {.Title = "The Breakfast Club", .Director = "John Hughes", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("2/15/1985"),
.Runtime = 97}, _
          New Movie With {.Title = "Pretty in Pink", .Director = "Howard Deutch", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("2/28/1986"),
.Runtime = 96}, _
          New Movie With {.Title = "Weird Science", .Director = "John Hughes", _
                .Genre = 0, .ReleaseDate = DateTime.Parse("8/2/1985"),
.Runtime = 94}, _
          New Movie With {.Title = "Breakfast at Tiffany's", .Director =
"Blake Edwards", _
                .Genre = 1, .ReleaseDate = DateTime.Parse("10/5/1961"),
.Runtime = 115}, _
          New Movie With {.Title = "The Graduate", .Director = "Mike Nichols", _
               .Genre = 1, .ReleaseDate = DateTime.Parse("4/2/1968"),
.Runtime = 105}, _
          New Movie With {.Title = "Dazed and Confused", .Director = "Richard
Linklater", _
                .Genre = 0, .ReleaseDate = DateTime.Parse("9/24/1993"),
.Runtime = 103}, _
          New Movie With {.Title = "Arthur", .Director = "Steve Gordon", _
               .Genre = 1, .ReleaseDate = DateTime.Parse("9/25/1981"),
.Runtime = 97}, _
          New Movie With {.Title = "Monty Python and the Holy Grail", _
               .Director = "Terry Gilliam", .Genre = 0, _
               .ReleaseDate = DateTime.Parse("5/10/1975"),
.Runtime = 91}, _
            New Movie With {.Title = "Dirty Harry", .Director = "Don Siegel", _
               .Genre = 2, .ReleaseDate = DateTime.Parse("12/23/1971"),
.Runtime = 102} _
}

        Return New List(Of Movie)(movies)
    End Function
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

Continued

```
    <title>My Favorite Movies</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:GridView ID="GridView1" runat="server">
        </asp:GridView>
    </div>
    </form>
</body>
</html>
```

C#

```
<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        var movies = GetMovies();

        this.GridView1.DataSource = movies;
        this.GridView1.DataBind();
    }

    public List<Movie> GetMovies()
    {
        return new List<Movie> {
            new Movie { Title="Shrek", Director="Andrew Adamson", Genre=0,
               ReleaseDate=DateTime.Parse("5/16/2001"), RunTime=89 } ,
            new Movie { Title="Fletch", Director="Michael Ritchie", Genre=0,
               ReleaseDate=DateTime.Parse("5/31/1985"), RunTime=96 } ,
            new Movie { Title="Casablanca", Director="Michael Curtiz", Genre=1,
               ReleaseDate=DateTime.Parse("1/1/1942"), RunTime=102 } ,
            new Movie { Title="Batman", Director="Tim Burton", Genre=1,
               ReleaseDate=DateTime.Parse("6/23/1989"), RunTime=126 } ,
            new Movie { Title="Dances with Wolves", Director="Kevin Costner",Genre=1,
               ReleaseDate=DateTime.Parse("11/21/1990"), RunTime=180 } ,
            new Movie { Title="Dirty Dancing", Director="Emile Ardolino", Genre=1,
               ReleaseDate=DateTime.Parse("8/21/1987"), RunTime=100 } ,
            new Movie { Title="The Parent Trap", Director="Nancy Meyers", Genre=0,
               ReleaseDate=DateTime.Parse("7/29/1998"), RunTime=127 } ,
            new Movie { Title="Ransom", Director="Ron Howard", Genre=1,
               ReleaseDate=DateTime.Parse("11/8/1996"), RunTime=121 } ,
            new Movie { Title="Ocean's Eleven", Director="Steven
Soderbergh", Genre=1,
               ReleaseDate=DateTime.Parse("12/7/2001"), RunTime=116 } ,
            new Movie { Title="Steel Magnolias", Director="Herbert Ross", Genre=1,
               ReleaseDate=DateTime.Parse("11/15/1989"), RunTime=117 } ,
            new Movie { Title="Mystic Pizza", Director="Donald Petrie", Genre=1,
               ReleaseDate=DateTime.Parse("10/21/1988"), RunTime=104 } ,
            new Movie { Title="Pretty Woman", Director="Garry Marshall", Genre=1,
               ReleaseDate=DateTime.Parse("3/23/1990"), RunTime=119 } ,
            new Movie { Title="Interview with the Vampire",
               Director="Neil Jordan", Genre=1,
               ReleaseDate=DateTime.Parse("11/11/1994"), RunTime=123 } ,
```

Continued

```
            new Movie { Title="Top Gun", Director="Tony Scott", Genre=2,
               ReleaseDate=DateTime.Parse("5/16/1986"), RunTime=110 } ,
            new Movie { Title="Mission Impossible", Director="Brian De Palma", Genre=2,
               ReleaseDate=DateTime.Parse("5/22/1996"), RunTime=110 } ,
            new Movie { Title="The Godfather", Director="Francis Ford Coppola",
              Genre=1, ReleaseDate=DateTime.Parse("3/24/1972"), RunTime=175 } ,
            new Movie { Title="Carlito's Way", Director="Brian De Palma",
              Genre=1, ReleaseDate=DateTime.Parse("11/10/1993"), RunTime=144 } ,
            new Movie { Title="Robin Hood: Prince of Thieves",
Director="Kevin Reynolds",
              Genre=1, ReleaseDate=DateTime.Parse("6/14/1991"), RunTime=143 } ,
            new Movie { Title="The Haunted", Director="Robert Mandel",
              Genre=1, ReleaseDate=DateTime.Parse("5/6/1991"), RunTime=100 } ,
            new Movie { Title="Old School", Director="Todd Phillips",
              Genre=0, ReleaseDate=DateTime.Parse("2/21/2003"), RunTime=91 } ,
            new Movie { Title="Anchorman: The Legend of Ron Burgundy",
              Director="Adam McKay", Genre=0,
              ReleaseDate=DateTime.Parse("7/9/2004"), RunTime=94 } ,
            new Movie { Title="Bruce Almighty", Director="Tom Shadyac",
              Genre=0, ReleaseDate=DateTime.Parse("5/23/2003"), RunTime=101 } ,
            new Movie { Title="Ace Ventura: Pet Detective", Director="Tom Shadyac",
              Genre=0, ReleaseDate=DateTime.Parse("2/4/1994"), RunTime=86 } ,
            new Movie { Title="Goonies", Director="Richard Donner",
              Genre=0, ReleaseDate=DateTime.Parse("6/7/1985"), RunTime=114 } ,
            new Movie { Title="Sixteen Candles", Director="John Hughes",
              Genre=1, ReleaseDate=DateTime.Parse("5/4/1984"), RunTime=93 } ,
            new Movie { Title="The Breakfast Club", Director="John Hughes",
              Genre=1, ReleaseDate=DateTime.Parse("2/15/1985"), RunTime=97 } ,
            new Movie { Title="Pretty in Pink", Director="Howard Deutch",
              Genre=1, ReleaseDate=DateTime.Parse("2/28/1986"), RunTime=96 } ,
            new Movie { Title="Weird Science", Director="John Hughes",
              Genre=0, ReleaseDate=DateTime.Parse("8/2/1985"), RunTime=94 } ,
            new Movie { Title="Breakfast at Tiffany's", Director="Blake Edwards",
              Genre=1, ReleaseDate=DateTime.Parse("10/5/1961"), RunTime=115 } ,
            new Movie { Title="The Graduate", Director="Mike Nichols",
              Genre=1, ReleaseDate=DateTime.Parse("4/2/1968"), RunTime=105 } ,
            new Movie { Title="Dazed and Confused", Director="Richard Linklater",
              Genre=0, ReleaseDate=DateTime.Parse("9/24/1993"), RunTime=103 } ,
            new Movie { Title="Arthur", Director="Steve Gordon",
              Genre=1, ReleaseDate=DateTime.Parse("9/25/1981"), RunTime=97 } ,
            new Movie { Title="Monty Python and the Holy Grail",
Director="Terry Gilliam",
              Genre=0, ReleaseDate=DateTime.Parse("5/10/1975"), RunTime=91 } ,
            new Movie { Title="Dirty Harry", Director="Don Siegel",
              Genre=2, ReleaseDate=DateTime.Parse("12/23/1971"), RunTime=102 }
        };
    }
</script>
```

Running the sample generates a typical ASP.NET Web page that includes a simple grid showing all of the Movie data on it.

Now, what happens when you want to start performing queries on the list of movies? For example, you might want to filter this data to show only a specific genre of movie. Listing 9-3 shows a typical way you might perform this filtering.

Listing 9-3: Filtering the listing Movie objects

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim movies = GetMovies()

    Dim query As New List(Of Movie)()
    For Each m In movies
        If (m.Genre = 0) Then
            query.Add(m)
        End If
    Next

    Me.GridView1.DataSource = query
    Me.GridView1.DataBind()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = new List<Movie>();
    foreach (var m in movies)
    {
        if (m.Genre == 0) query.Add(m);
    }

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

As this sample shows, to filter the data so that the page displays Movies in a specific genre only requires the creation of a new temporary collection and the use of a `foreach` loop to iterate through the data.

While this technique seems easy enough, it still requires that you define what you want done (find all movies in the genre), and also that you explicitly define how it should be done (use a temporary collection and a `foreach` loop). Additionally, what happens when you need to perform more complex queries, involving grouping or sorting? Now the complexity of the code dramatically increases, as shown in Listing 9-4.

Listing 9-4: Grouping and sorting the List of Movie objects

VB

```
Public Class Grouping
    Private _genre As Integer
    Private _movieCount As Integer

    Public Property Genre() As Integer
        Get
            Return _genre
```

Continued

```
        End Get
        Set(ByVal value As Integer)
            _genre = value
        End Set
    End Property

    Public Property MovieCount() As Integer
        Get
            Return _movieCount
        End Get
        Set(ByVal value As Integer)
            _movieCount = value
        End Set
    End Property
End Class

Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim movies = GetMovies()

    Dim groups As New Dictionary(Of String, Grouping)

    For Each m In movies

        If (Not groups.ContainsKey(m.Genre)) Then
            groups(m.Genre) = _
                New Grouping With {.Genre = m.Genre, .MovieCount = 0}
        End If

        groups(m.Genre).MovieCount = groups(m.Genre).MovieCount + 1
    Next

    Dim results As New List(Of Grouping)(groups.Values)
    results.Sort(AddressOf MovieSort)

    Me.GridView1.DataSource = results
    Me.GridView1.DataBind()
End Sub

Private Function MovieSort(ByVal x As Grouping, ByVal y As Grouping) As Integer
    Return IIf(x.MovieCount > y.MovieCount, -1, IIf(x.MovieCount
< y.MovieCount, 1, 0))
End Function
```

C#

```
public class Grouping
{
    public int Genre { get; set; }
    public int MovieCount { get; set; }
}

protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();
```

Continued

```
        Dictionary<int, Grouping> groups = new Dictionary<int, Grouping>();
        foreach (Movie m in movies)
        {
            if (!groups.ContainsKey(m.Genre))
            {
                groups[m.Genre] = new Grouping { Genre = m.Genre, MovieCount = 0 };
            }
            groups[m.Genre].MovieCount++;
        }

        List<Grouping> results = new List<Grouping>(groups.Values);
        results.Sort(delegate(Grouping x, Grouping y)
        {
            return
                x.MovieCount > y.MovieCount ? -1 :
                x.MovieCount < y.MovieCount ? 1 :
                0;
        });

        this.GridView1.DataSource = results;
        this.GridView1.DataBind();
    }
```

To group the Movie data into genres and count how many movies are in each genre requires the addition of a new class, the creation of a Dictionary, and the implementation of a delegate, all fairly complex requirements for such a seemingly simple task, and again, not only defining very specifically what you want done, but very explicitly how it should be done.

Additionally, because the complexity of the code increases so much, it becomes quite difficult to actually determine what this code is doing. Consider this: What if you were asked to modify this code in an existing application that you were unfamiliar with? How long would it take you to figure out what it was doing?

Replacing Traditional Queries with LINQ

LINQ was created to address many of the shortcomings of querying collections of data that were discussed in the previous section. Rather than requiring you to very specifically define exactly how you want a query to execute, LINQ gives you the power to stay at a more abstract level. By simply defining what you want the query to return, you leave it up to .NET and its compilers to determine the specifics of exactly how the query will be run.

In the preceding section, you looked at the current state of object querying with today's .NET languages. In this section, let's take a look at LINQ and see how using it can greatly simplify these queries, as well as other types of queries. To do this, the samples in this section start out by simply modifying the samples from the previous section to show you how easy LINQ makes the same tasks.

Before you get started, understand that LINQ is an extension to .NET, and because of this, is isolated in its own set of assemblies. The base LINQ functionality is located in the new `System.Core.dll` assembly. This assembly does not replace any existing framework functionality, but simply augments it.

Additionally, by default, projects in Visual Studio 2008 include a reference to this assembly so when starting a new ASP.NET Web project, LINQ should be readily available to you.

Basic LINQ Queries and Projections

In order to start modifying the prior sections samples to using LINQ queries, you first need to add the LINQ namespace to the Web page, as shown in Listing 9-5.

Listing 9-5: Adding the LINQ namespace

```
<%@ Import Namespace="System.Linq" %>
```

Adding this namespace gives the page access to all of the basic LINQ functionality. If you are using the code-behind development model, then the LINQ namespace should already be included in your code-behind class.

> *Note that the default web.config file included with a Visual Basic Web site already includes the System.Linq namespace declaration in it, so if you are using this type of project you do not need to manually add the namespace.*

Next, you can start by modifying code from Listing 9-2. If you remember, this basic sample simply generates a generic List of movies and binds the list to a GridView control. Listing 9-6 shows how the code can be modified to use LINQ to query the movies list and bind the resultset to the GridView.

Listing 9-6: Creating a query with LINQ

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim movies = GetMovies()

    Dim query = From m In movies _
                Select m

    Me.GridView1.DataSource = query
    Me.GridView1.DataBind()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = from m in movies
                select m;

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

If we deconstruct the code sample, there are three basic actions happening. First, the code uses the `GetMovies()` method to obtain the generic `List<Movie>` collection.

Next, the code uses a very simple LINQ query to select all of the Movie objects from the generic movies collection. Notice that this specific LINQ query utilizes new language keywords like `from` and `select` in the query statement. These syntax additions are first class members of the .NET languages, therefore Visual Studio 2008 can offer you development assistance such as strong type checking and IntelliSense, making it easier for you to find and fix problems in your code.

The query also defines a new variable `m`. This variable is used in two ways in the query. First, by defining it in the `from` statement from `m`, we are telling LINQ to make `m` represent the individual collection item, which in this case is a `Movie` object. Telling LINQ this enables it to understand the structure of the objects we are querying, and as you will see later, also gives us IntelliSense to help create the query.

The second use of `m` in the query is in the `select` statement. Using `m` in the `select` statement tells LINQ to output a projection that matches the structure of `m`. In this case that means LINQ creates a projection that matches the `Movie` object structure.

We could just have easily created our own custom projection by explicitly defining the fields we wanted returned from the query using the new keyword along with the select operator. This is shown below in Listing 9-7.

Listing 9-7: Creating a custom projection with LINQ

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim movies = GetMovies()

    Dim query = From m In movies _
                Select New With {m.Title, m.Genre}

    Me.GridView1.DataSource = query
    Me.GridView1.DataBind()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = from m in movies
                select new { m.Title, m.Genre };

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

Notice that rather than simply selecting `m`, we have defined a new projection containing only the Title and Genre values.

You can even go so far as to explicitly define the field names that the objects in the resultset will expose. For example, you may want to more explicitly name the Title and Genre fields to more fully describe their contents. Using LINQ, it’s easy to do this, as shown in Listing 9-8.

Listing 9-8: Creating custom projection field names

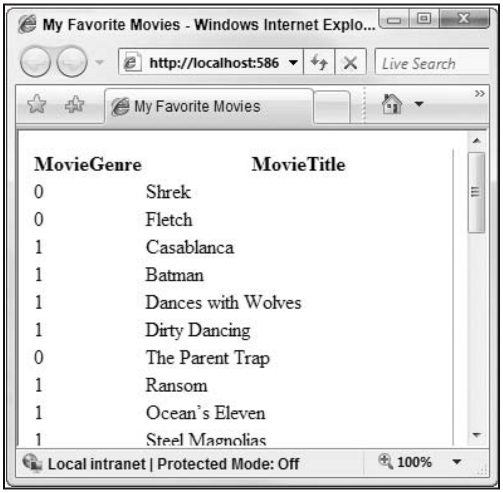
VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim movies = GetMovies()

    Dim query = From m In movies _
                Select New With {.MovieTitle = m.Title, .MovieGenre = m.Genre}

    Me.GridView1.DataSource = query
    Me.GridView1.DataBind()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = from m in movies
                select new { MovieTitle = m.Title, MovieGenre = m. Genre };

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

This sample explicitly defined the Fields that will be exposed by the resultset as MovieTitle and MovieGenre. You can see in Figure 9-1, that because of this change, the column headers in the GridView have changed to match.

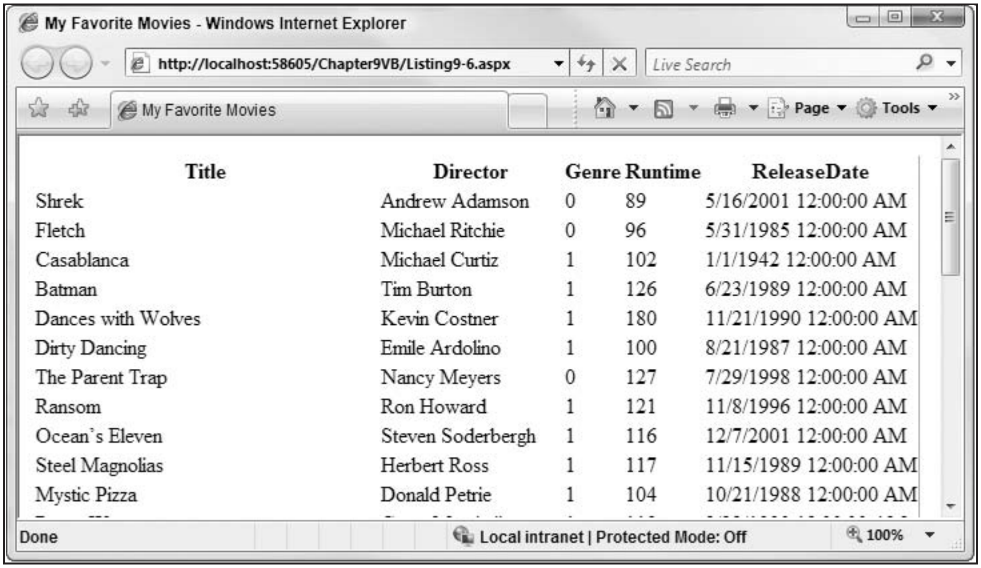

My Favorite Movies - Windows Internet Explo...
http://localhost:586
Live Search
My Favorite Movies
Local intranet | Protected Mode: Off
100%

MovieGenre	MovieTitle
0	Shrek
0	Fletch
1	Casablanca
1	Batman
1	Dances with Wolves
1	Dirty Dancing
0	The Parent Trap
1	Ransom
1	Ocean's Eleven
1	Steel Magnolias

Figure 9-1 Customized GridView column headers as the result of the LINQ projection

Finally the code binds the GridView control to the enumerable list of Movie object returned by the LINQ query.

As shown in Figure 9-2, running the code from Listing 9-6 results in the same vanilla Web page as the one generated by Listing 9-2.


My Favorite Movies - Windows Internet Explorer
http://localhost:58605/Chapter9VB/Listing9-6.aspx
Live Search
My Favorite Movies
Page
Tools
Done
Local intranet | Protected Mode: Off
100%

Title	Director	Genre	Runtime	ReleaseDate
Shrek	Andrew Adamson	0	89	5/16/2001 12:00:00 AM
Fletch	Michael Ritchie	0	96	5/31/1985 12:00:00 AM
Casablanca	Michael Curtiz	1	102	1/1/1942 12:00:00 AM
Batman	Tim Burton	1	126	6/23/1989 12:00:00 AM
Dances with Wolves	Kevin Costner	1	180	11/21/1990 12:00:00 AM
Dirty Dancing	Emile Ardolino	1	100	8/21/1987 12:00:00 AM
The Parent Trap	Nancy Meyers	0	127	7/29/1998 12:00:00 AM
Ransom	Ron Howard	1	121	11/8/1996 12:00:00 AM
Ocean's Eleven	Steven Soderbergh	1	116	12/7/2001 12:00:00 AM
Steel Magnolias	Herbert Ross	1	117	11/15/1989 12:00:00 AM
Mystic Pizza	Donald Petrie	1	104	10/21/1988 12:00:00 AM

Figure 9-2 The results of a basic LINQ query bound to a GridView control

LINQ also includes the ability to order the results using the order by statement. As with SQL you can choose to order the results in either ascending or descending order, as shown in Listing 9-9.

Listing 9-9: Controlling data ordering using LINQ

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim movies = GetMovies()

    Dim query = From m In movies _
                Order By m.Title Descending _
                Select New With {.MovieTitle = m.Title, .MovieGenre = m.Genre}

    Me.GridView1.DataSource = query
    Me.GridView1.DataBind()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();
```

```
        var query = from m in movies
                    orderby m.Title descending
                    select new { MovieTitle = m.Title, MovieGenre = m. Genre };

        this.GridView1.DataSource = query;
        this.GridView1.DataBind();
    }
```

Another great feature of the new LINQ syntax is the dramatic improvement in readability and understandability that it makes in your code. LINQ enables you to simply express the intention of your query, indicating to the compiler what you want your code to do, but leaving it up to the compiler to best determine how it should be done.

While these new keywords are what enable you to construct LINQ queries using a simple and clear SQL-like syntax, rest assured there is no magic occurring. These keywords actually map to extension methods on the Movies collection. You could actually write the same LINQ query directly using these extension methods and it would look like this:

VB

```
Dim query = movies.Select( Function(m as Movie) m )
```

C#

```
var query = movies.Select(m => m);
```

This is what the compiler translates the keyword syntax into during its compilation process. You may be wondering how the `Select` *method got added to our generic* `List<Movies>` *collection because if you look at the object structure of* `List<T>`*, there is no* `Select` *method. LINQ adds the* `Select` *method, and many other methods it uses to the base Enumerable class, using Extension Methods. Therefore, any class that implements IEnumerable will be extended by LINQ with these methods. You can see all of the methods added by LINQ by right-clicking on the* `Select` *method in Visual Studio and choosing the View Definition option from the context menu. Doing this causes Visual Studio to display the class metadata for LINQ's* `Enumerable` *class. If you scroll through this class, you will see not only* `Select`*, but other methods such as* `Where`*,* `Count`*,* `Min`*,* `Max`*, and many other methods that LINQ automatically adds to any object that implements the IEnumerable interface.*

Delayed Execution

An interesting feature of LINQ is its delayed execution behavior. This means that even though you may execute the query statements at a specific point in your code, LINQ is smart enough to delay the actual execution of the query until it is accessed. For example, in the previous samples, although the LINQ query was written before the binding of the GridView controls, LINQ will not actually execute the query we have defined until the GridView control begins to enumerate through the query results.

Query Filters

LINQ also supports adding query filters using a familiar SQL-like `where` syntax. You can modify the LINQ query from Listing 9-3 to add filtering capabilities by adding a `where` clause to the query, as shown in Listing 9-10.

Listing 9-10: Adding a filter to a LINQ query

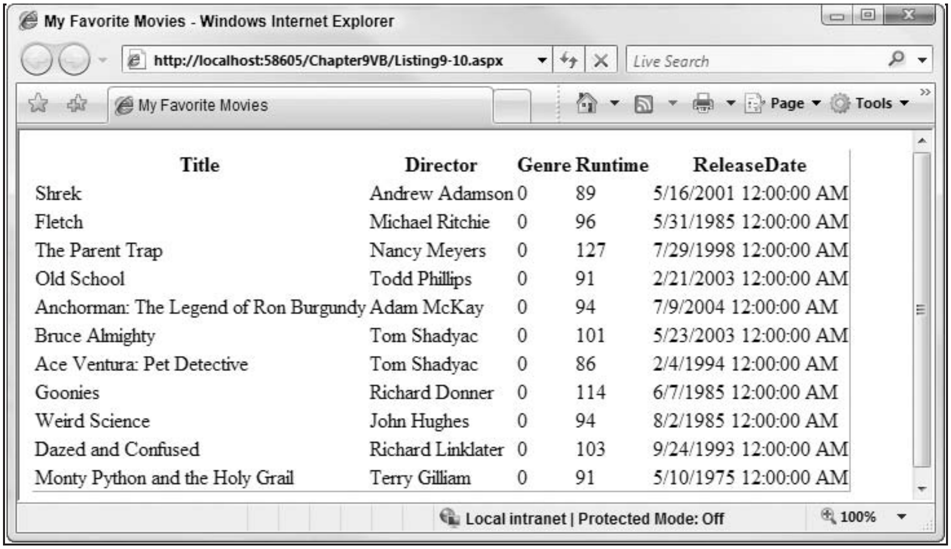
VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim movies = GetMovies()

    Dim query = From m In movies _
                Where m.Genre = 0 _
                Select m

    Me.GridView1.DataSource = query
    Me.GridView1.DataBind()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = from m in movies
                where m.Genre==0
                select m;

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

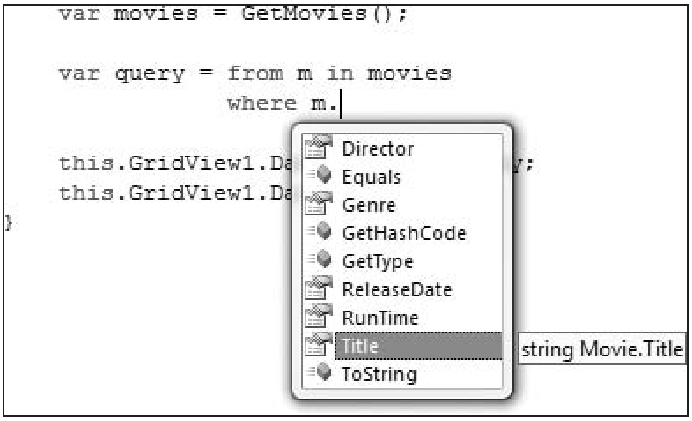
By adding this simple `where` clause to the LINQ query, the results returned by the query are filtered to show movies from the 0 genre only, as shown in Figure 9-3.


My Favorite Movies - Windows Internet Explorer
http://localhost:58605/Chapter9VB/Listing9-10.aspx
Live Search
My Favorite Movies
Page
Tools
Local intranet | Protected Mode: Off
100%

Title	Director	Genre	Runtime	ReleaseDate
Shrek	Andrew Adamson	0	89	5/16/2001 12:00:00 AM
Fletch	Michael Ritchie	0	96	5/31/1985 12:00:00 AM
The Parent Trap	Nancy Meyers	0	127	7/29/1998 12:00:00 AM
Old School	Todd Phillips	0	91	2/21/2003 12:00:00 AM
Anchorman: The Legend of Ron Burgundy	Adam McKay	0	94	7/9/2004 12:00:00 AM
Bruce Almighty	Tom Shadyac	0	101	5/23/2003 12:00:00 AM
Ace Ventura: Pet Detective	Tom Shadyac	0	86	2/4/1994 12:00:00 AM
Goonies	Richard Donner	0	114	6/7/1985 12:00:00 AM
Weird Science	John Hughes	0	94	8/2/1985 12:00:00 AM
Dazed and Confused	Richard Linklater	0	103	9/24/1993 12:00:00 AM
Monty Python and the Holy Grail	Terry Gilliam	0	91	5/10/1975 12:00:00 AM

Figure 9-3 A filtered list of Movies

Also, notice that, because LINQ is a first-class member of .NET, Visual Studio is able to provide an excellent coding experience as you are constructing your LINQ queries. In this sample, as you enter the `where` clause, Visual Studio gives you IntelliSense for the possible parameters of `m` (the Movie object), as shown in Figure 9-4.



```
    var movies = GetMovies();

    var query = from m in movies
                where m.

    this.GridView1.Da                    y;
    this.GridView1.Da
}
```

Figure 9-4 Because LINQ is a first class language concept, Visual Studio can give you Intellisense

The `where` clause in LINQ behaves similarly to the SQL `where` clause, enabling you to include sub-queries and multiple where clauses, as shown in Listing 9-11.

Listing 9-11: Adding a Where clause to a LINQ query

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim movies = GetMovies()

    Dim query = From m In movies _
                Where m.Genre = 0 And m.Runtime > 92 _
                Select m

    Me.GridView1.DataSource = query
    Me.GridView1.DataBind()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = from m in movies
                where m.Genre == 0 && m.RunTime > 92
                select m;

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

In this sample, the `where` clause includes two parameters, one restricting the movie genre, the other restricting the movie’s runtime.

Data Grouping

LINQ also greatly simplifies grouping data, again using a SQL-like `group` syntax. To show how easy LINQ makes this, you can modify the original Listing 9-4 to use a LINQ query. The modified code is shown in Listing 9-12.

Listing 9-12: Grouping data using a LINQ query

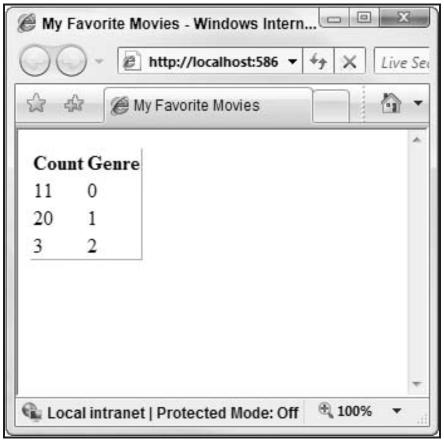
VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim movies = GetMovies()

    Dim query = From m In movies _
                Group By m.Genre Into g = Group, Count()

    Me.GridView1.DataSource = query
    Me.GridView1.DataBind()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    var query = from m in movies
                group m by m.Genre into g
                select new { Genre = g.Key, Count = g.Count() };

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

This LINQ query uses the `group` keyword to group the movie data by genre. Additionally, because a group action does not naturally result in any output, the query creates a custom query projection using the techniques discussed earlier. The results of this query are shown in Figure 9-5.


My Favorite Movies - Windows Intern...
http://localhost:586
Live Se
My Favorite Movies
Local intranet | Protected Mode: Off
100%

Count	Genre
11	0
20	1
3	2

Figure 9-5 Grouped data results

Using LINQ to do this allows you to significantly reduce the lines of code required. If we compare the amount of code required to perform the grouping action in Listing 9-4, with the previous listing using LINQ, you can see that the number of lines of code has dropped from 18 to 3, and the readability and clarity of the code has improved.

Other LINQ Operators

Besides basic selection, filtering and grouping, LINQ also includes many operators you can execute on enumerable objects. Most of these operators are available for you to use and are similar to operators you find in SQL, such as Count, Min, Max, Average, and Sum, as shown in Listing 9-13.

Listing 9-13: Using LINQ query operators

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim movies = GetMovies()

    Me.TotalMovies.Text = movies.Count.ToString()
    Me.LongestRuntime.Text = movies.Max(Function(m) m.Runtime).ToString()
    Me.ShortestRuntime.Text = movies.Min(Function(m) m.Runtime).ToString()
    Me.AverageRuntime.Text = movies.Average(Function(m) m.Runtime).ToString()

End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();

    this.TotalMovies.Text = movies.Count.ToString();
    this.LongestRuntime.Text = movies.Max(m => m.RunTime).ToString();
    this.ShortestRuntime.Text = movies.Min(m => m.RunTime).ToString();
    this.AverageRuntime.Text = movies.Average(m => m.RunTime).ToString();
}
```

This listing demonstrates the use of the Count, Max, Min, and Average operators with the movies collection. Notice that for all but the Count operator, you need to provide the method with the specific field you want to execute the operation on. This is done using a Lambda expression.

LINQ Joins

LINQ also supports the unioning of data from different collections using a familiar SQL-like join syntax. For example, in our sample data thus far, we have only been able to display the Genre as a numeric ID. It would be preferable to actually display the name of each Genre instead. To do this, you simply create a Genre class, which defines the properties of the genre, as shown in Listing 9-14

Listing 9-14: A simple Genre class

VB

```
Public Class Genre
    Private _id As Integer
    Private _name As String
```

Continued

```
    Public Property ID() As Integer
        Get
            Return _id
        End Get
        Set(ByVal value As Integer)
            _id = value
        End Set
    End Property

    Public Property Name() As String
        Get
            Return _name
        End Get
        Set(ByVal value As String)
            _name = value
        End Set
    End Property
End Class
```

C#

```
public class Genre
{
    public int ID { get; set; }
    public string Name { get; set; }
}
```

Next you can add a `GetGenres` method to your Web page that returns a list of Genre objects, as shown in Listing 9-15.

Listing 9-15: Populating a collection of Genres

VB

```
Public Function GetGenres() As List(Of Genre)
    Dim genres As Genre() = { _
        New Genre With {.ID = 0, .Name = "Comedy"}, _
        New Genre With {.ID = 1, .Name = "Drama"}, _
        New Genre With {.ID = 2, .Name = "Action"} _
    }

    Return New List(Of Genre)(genres)
End Function
```

C#

```
public List<Genre> GetGenres()
{
    return new List<Genre> {
                new Genre { ID=0, Name="Comedy" } ,
                new Genre { ID=1, Name="Drama" } ,
                new Genre { ID=2, Name="Action" }
            };
}
```

Finally, you can modify the Page Load event, including the LINQ query, to retrieve the Genres list and, using LINQ, join that to the Movies list. This is shown in Listing 9-16.

Listing 9-16: Joining Genre data with Movie data using a LINQ query

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim movies = GetMovies()
    Dim genres = GetGenres()

    Dim query = From m In movies Join g In genres _
                On m.Genre Equals g.ID _
                Select New With {.Title = m.Title, .Genre = g.Name}

    GridView1.DataSource = query
    GridView1.DataBind()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();
    var genres = GetGenres();

    var query = from m in movies
                join g in genres on m.Genre equals g.ID
                select new { m.Title, Genre = g.Name } ;

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

As you can see in this sample, the join syntax is relatively simple. You tell LINQ to include the genres object, and then tell LINQ which fields it should associate.

Paging Using LINQ

LINQ also makes it much easier to include paging logic in your Web application by exposing the `Skip` and `Take` methods. The `Skip` method enables you to skip a defined number of records in the resultset. The `Take` method enables you to specify the number of records to return from the resultset. By calling `Skip`, and then `Take`, you can return a specific number of records from a specific location of the resultset. This is shown in Listing 9-17.

Listing 9-17: Simple Paging using LINQ methods

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim movies = GetMovies()
    Dim genres = GetGenres()

    Dim query = (From m In movies _
                 Join g In genres On m.Genre Equals g.ID _
```

Continued

```
                    Select New With {m.Title, .Genre = g.Name}).Skip(10).Take(10)

        Me.GridView1.DataSource = query
        Me.GridView1.DataBind()
        End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    var movies = GetMovies();
    var genres = GetGenres();

    var query = (from m in movies
                join g in genres on m.Genre equals g.ID
                select new { m.Title, g.Name }).Skip(10).Take(10);

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

When running this code, you will see that the results start with the tenth record in the list, and only ten records are displayed.

LINQ to XML

The second flavor of LINQ is called LINQ to XML (or XLINQ). As the name implies, LINQ to XML enables you to use the same basic LINQ syntax to query XML documents. As with the basic LINQ features, the LINQ to XML features of .NET are included as an extension to the basic .NET framework, and do not change any existing functionality. Also, as with the core LINQ features, the LINQ to XML features are contained in their own separate assembly, the System.Xml.Linq assembly.

In this section, to show how you can use LINQ to query XML, we use the same basic Movie data as in the previous section, but converted to XML. Listing 9-18 shows a portion of the Movie data converted to a simple XML document. The XML file containing the complete set of converted data can be found in the downloadable code for this chapter.

Listing 9-18: Sample Movies XML data file

```
<?xml version="1.0" encoding="utf-8" ?>
<Movies>
  <Movie>
    <Title>Shrek</Title>
    <Director>Andrew Adamson</Director>
    <Genre>0</Genre>
    <ReleaseDate>5/16/2001</ReleaseDate>
    <RunTime>89</RunTime>
  </Movie>
  <Movie>
    <Title>Fletch</Title>
    <Director>Michael Ritchie</Director>
    <Genre>0</Genre>
    <ReleaseDate>5/31/1985</ReleaseDate>
    <RunTime>96</RunTime>
  </Movie>
  <Movie>
    <Title>Casablanca</Title>
    <Director>Michael Curtiz</Director>
    <Genre>1</Genre>
    <ReleaseDate>1/1/1942</ReleaseDate>
    <RunTime>102</RunTime>
  </Movie>
</Movies>
```

To get started seeing how you can use LINQ to XML to query XML documents, let's walk through some of the same basic queries we started with in the previous section. Listing 9-19 demonstrates a simple selection query using LINQ to XML.

Listing 9-19: Querying the XML Data file using LINQ

VB

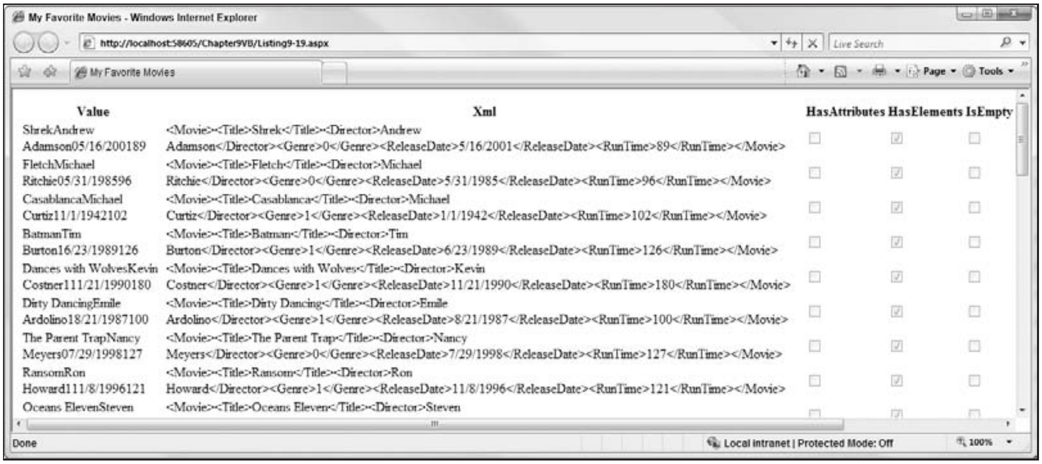
```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
       Dim query = From m In XElement.Load(MapPath("Movies.xml")).Elements("Movie") _
                    Select m

        Me.GridView1.DataSource = query
        Me.GridView1.DataBind()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>My Favorite Movies</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:GridView ID="GridView1" runat="server">
        </asp:GridView>
    </div>
    </form>
</body>
</html>
```

C#

```
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        var query = from m in XElement.Load(MapPath("Movies.xml")).Elements("Movie")
                    select m;

        this.GridView1.DataSource = query;
        this.GridView1.DataBind();
    }
</script>
```

Notice that in this query, you tell LINQ directly where to load the XML data from, and from which elements in that document it should retrieve the data, which in this case are all of the Movie elements. Other than that minor change, the LINQ query is identical to queries we have seen previously.

When you execute this code, you get a page that looks like Figure 9-6.


My Favorite Movies - Windows Internet Explorer
http://localhost:58605/Chapter9VB/Listing9-19.aspx
Live Search
My Favorite Movies
Page
Tools
Done
Local intranet | Protected Mode: Off
100%

Value	Xml	HasAttributes	HasElements	IsEmpty
ShrekAndrew Adamson05/16/200189	<Movie><Title>Shrek</Title><Director>Andrew Adamson</Director><Genre>0</Genre><ReleaseDate>5/16/2001</ReleaseDate><RunTime>89</RunTime></Movie>	☐	☑	☐
FletchMichael Ritchie05/31/198596	<Movie><Title>Fletch</Title><Director>Michael Ritchie</Director><Genre>0</Genre><ReleaseDate>5/31/1985</ReleaseDate><RunTime>96</RunTime></Movie>	☐	☑	☐
CasablancaMichael Curtiz11/1/1942102	<Movie><Title>Casablanca</Title><Director>Michael Curtiz</Director><Genre>1</Genre><ReleaseDate>1/1/1942</ReleaseDate><RunTime>102</RunTime></Movie>	☐	☑	☐
BatmanTim Burton16/23/1989126	<Movie><Title>Batman</Title><Director>Tim Burton</Director><Genre>1</Genre><ReleaseDate>6/23/1989</ReleaseDate><RunTime>126</RunTime></Movie>	☐	☑	☐
Dances with WolvesKevin Costner111/21/1990180	<Movie><Title>Dances with Wolves</Title><Director>Kevin Costner</Director><Genre>1</Genre><ReleaseDate>11/21/1990</ReleaseDate><RunTime>180</RunTime></Movie>	☐	☑	☐
Dirty DancingEmile Ardolino18/21/1987100	<Movie><Title>Dirty Dancing</Title><Director>Emile Ardolino</Director><Genre>1</Genre><ReleaseDate>8/21/1987</ReleaseDate><RunTime>100</RunTime></Movie>	☐	☑	☐
The Parent TrapNancy Meyers07/29/1998127	<Movie><Title>The Parent Trap</Title><Director>Nancy Meyers</Director><Genre>0</Genre><ReleaseDate>7/29/1998</ReleaseDate><RunTime>127</RunTime></Movie>	☐	☑	☐
RansomRon Howard111/8/1996121	<Movie><Title>Ransom</Title><Director>Ron Howard</Director><Genre>1</Genre><ReleaseDate>11/8/1996</ReleaseDate><RunTime>121</RunTime></Movie>	☐	☑	☐
Oceans ElevenSteven	<Movie><Title>Oceans Eleven</Title><Director>Steven	☐	☑	☐

Figure 9-6 LINQ to XML raw query results

Notice that the fields included in the resultset of the query don't really show the node data as you might have expected, with each child node as a separate Field in the GridView. This is because the query used in the Listing returns a collection of generic XElement objects, not Movie objects as you might have expected. This is because by itself, LINQ has no way of identifying what object type each node should be mapped to. Thankfully, you can add a bit of mapping logic to the query to tell it to map each node to a Movie object, and how the nodes sub-elements should map to the properties of the Movie object. This is shown in Listing 9-20.

Listing 9-20: Mapping XML elements using LINQ

VB

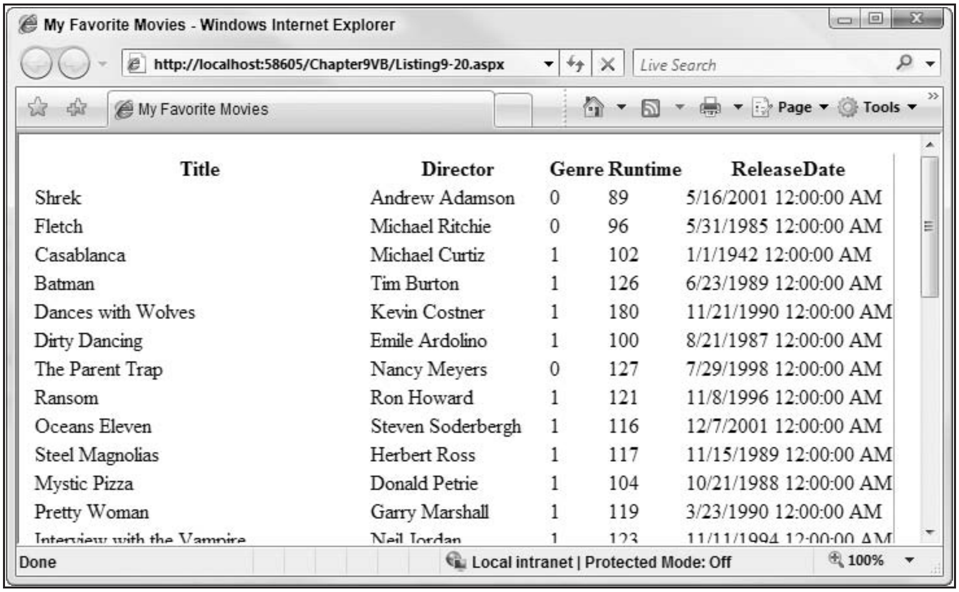
```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim query = From m In XElement.Load(MapPath("Movies.xml")).Elements("Movie") _
                Select New Movie With { _
                                        .Title = CStr(m.Element("Title")), _
                                        .Director = CStr(m.Element("Director")), _
                                        .Genre = CInt(m.Element("Genre")), _
                                        .ReleaseDate = CDate(m.Element
("ReleaseDate")), _
                                        .Runtime = CInt(m.Element("Runtime")) _
                                      }

    Me.GridView1.DataSource = query
    Me.GridView1.DataBind()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
```

```
{
    var query = from m in XElement.Load(MapPath("Movies.xml")).Elements("Movie")
                select new Movie {
                           Title = (string)m.Element("Title"),
                           Director = (string)m.Element("Director"),
                           Genre = (int)m.Element("Genre"),
                           ReleaseDate = (DateTime)m.Element("ReleaseDate"),
                           RunTime = (int)m.Element("RunTime")
                       };

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

As you can see, we have modified the query to include mapping logic so that LINQ knows what our actual intentions are — to create a resultset that contains the values of the Movie elements inner nodes. Running this code now results in a GridView that contains what we want, as shown in Figure 9-7.


My Favorite Movies - Windows Internet Explorer
http://localhost:58605/Chapter9VB/Listing9-20.aspx
Live Search
My Favorite Movies
Page
Tools
Done
Local intranet | Protected Mode: Off
100%

Title	Director	Genre	Runtime	ReleaseDate
Shrek	Andrew Adamson	0	89	5/16/2001 12:00:00 AM
Fletch	Michael Ritchie	0	96	5/31/1985 12:00:00 AM
Casablanca	Michael Curtiz	1	102	1/1/1942 12:00:00 AM
Batman	Tim Burton	1	126	6/23/1989 12:00:00 AM
Dances with Wolves	Kevin Costner	1	180	11/21/1990 12:00:00 AM
Dirty Dancing	Emile Ardolino	1	100	8/21/1987 12:00:00 AM
The Parent Trap	Nancy Meyers	0	127	7/29/1998 12:00:00 AM
Ransom	Ron Howard	1	121	11/8/1996 12:00:00 AM
Oceans Eleven	Steven Soderbergh	1	116	12/7/2001 12:00:00 AM
Steel Magnolias	Herbert Ross	1	117	11/15/1989 12:00:00 AM
Mystic Pizza	Donald Petrie	1	104	10/21/1988 12:00:00 AM
Pretty Woman	Garry Marshall	1	119	3/23/1990 12:00:00 AM
Interview with the Vampire	Neil Jordan	1	123	11/11/1994 12:00:00 AM

Figure 9-7 LINQ to XML query with the data properly mapped to a Movie object

Note that the XElements `Load` method attempts to load the entire XML document; therefore, it is not a good idea to try to load very large XML files using this method.

Joining XML Data

LINQ to XML supports all of the same query filtering and grouping operations as LINQ to Objects. It also supports joining data, and can actually union together data from two different XML documents — a task that previously would have been quite difficult. Let's look at the same basic join scenario as was presented in the LINQ to objects section. Again, our basic XML data includes only an ID value for the Genre. It would, however, be better to show the actual Genre name with our resultset.

In the case of the XML data, rather than being kept in a separate List, the Genre data is actually stored in a completed separate XML file, showing in Listing 9-21.

Listing 9-21: Genres XML data

```
<?xml version="1.0" encoding="utf-8" ?>
<Genres>
    <Genre>
        <ID>0</ID>
        <Name>Comedy</Name>
    </Genre>
    <Genre>
        <ID>1</ID>
        <Name>Drama</Name>
    </Genre>
    <Genre>
        <ID>2</ID>
        <Name>Action</Name>
    </Genre>
</Genres>
```

To join the data together, you can use a very similar join query to that used in Listing 9-16. This is shown in Listing 9-22.

Listing 9-22: Joining XML data using LINQ

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)

    Dim query = From m In XElement.Load(MapPath("Listing9-18.xml"))
.Elements("Movie") _
                Join g In XElement.Load(MapPath("Listing9-21.xml")).Elements
("Genre") _
                    On CInt(m.Element("Genre")) Equals CInt(g.Element("ID")) _
                Select New With { _
                    .Title = CStr(m.Element("Title")), _
                    .Director = CStr(m.Element("Director")), _
                    .Genre = CStr(g.Element("Name")), _
                    .ReleaseDate = CDate(m.Element("ReleaseDate")), _
                    .Runtime = CInt(m.Element("RunTime")) _
                }

    Me.GridView1.DataSource = query
    Me.GridView1.DataBind()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    var query = from m in XElement.Load(MapPath("Movies.xml")).Elements("Movie")
                join g in XElement.Load(MapPath("Genres.xml")).Elements("Genre")
                    on (int)m.Element("Genre") equals (int)g.Element("ID")
                select new {
```

```
                            Title = (string)m.Element("Title"),
                            Director = (string)m.Element("Director"),
                            Genre = (string)g.Element("Name"),
                            ReleaseDate = (DateTime)m.Element("ReleaseDate"),
                            RunTime = (int)m.Element("RunTime")
                        };

        this.GridView1.DataSource = query;
        this.GridView1.DataBind();
    }
```

In this sample, you can see we are using the `XElement.Load` method as part of the LINQ join statement to tell LINQ where to load the Genre data from. Once the data is joined, you can access the elements of the Genre data as you can the elements of the movie data.

LINQ to SQL

LINQ to SQL is the last form of LINQ in this release of .NET. LINQ to SQL, as the name implies, enables you to quickly and easily query SQL-based data sources, such as SQL Server 2005. As with the prior flavors of LINQ, LINQ to SQL is an extension of .NET. Its features are located in the System.Data.Linq assembly.

In addition to the normal Intellisense and strong type checking that every flavor of LINQ gives you, LINQ to SQL also includes a basic Object Relation (O/R) mapper directly in Visual Studio. The O/R mapper enables you to quickly map SQL-based data sources to CLR objects that you can then use LINQ to query. It is the easiest way to get started using LINQ to SQL.

The O/R mapper is used by adding the new Linq to SQL Classes file to your Web site project. The Linq to SQL File document type allows you to easily and visually create data contexts that you can then access and query with LINQ queries. Figure 9-8 shows the Linq to SQL Classes file type in the Add New Item dialog.

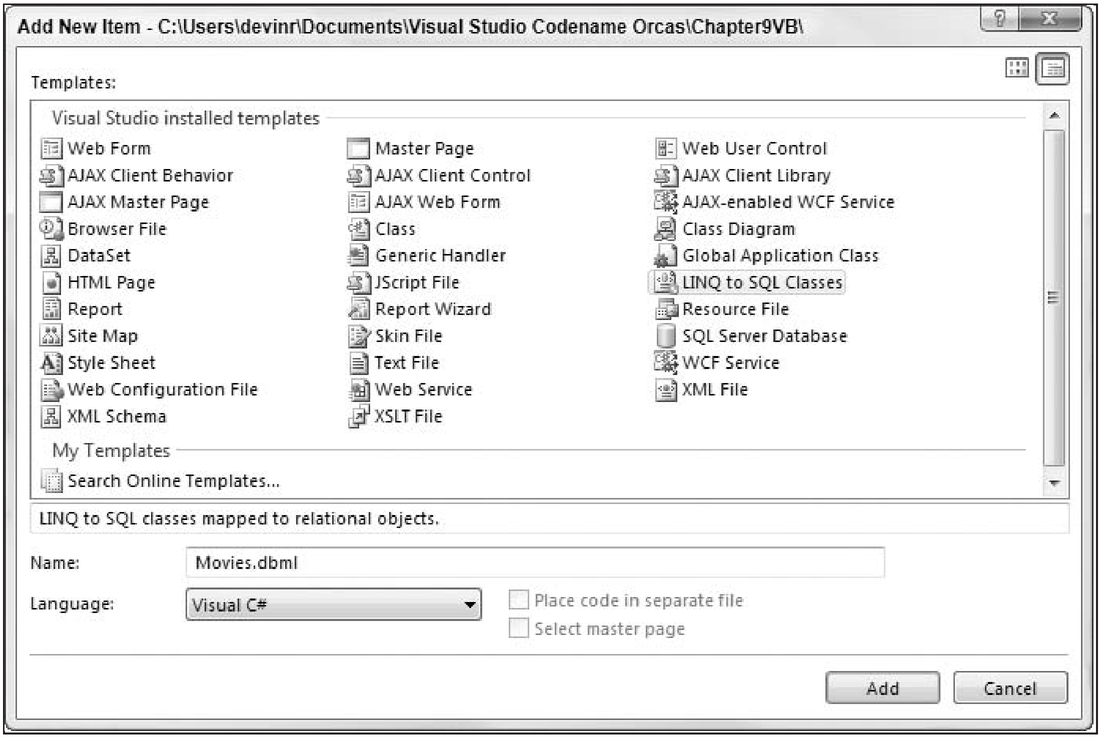
After clicking the Add New Items dialog's OK button to add the file to your project, Visual Studio notifies you that it wants to add the LINQ to SQL File to your Web site's App_Code directory. By locating the file there, the data context created by the LINQ to SQL Classes file will be accessible from anywhere in your Web site.

Once the file has been added, Visual Studio automatically opens it in the LINQ to SQL design surface. This is a simple Object Relation mapper design tool, enabling you to add, create, remove, and relate data objects. As you modify objects to the design surface, LINQ to SQL is generating object classes that mirror the structure of each of those objects. Later when you are ready to begin writing LINQ queries against the data objects, these classes will allow Visual Studio to provide you with design-time Intellisense support, strong typing and compile-time type checking. Because the O/R mapper is primarily designed to be used with LINQ to SQL, it also makes it easy to create CLR object representations of SQL objects, such as Tables, Views, and Stored Procedures.

To demonstrate using LINQ to SQL, we will use the same sample Movie data used in previous sections of this chapter. For this section, the data is stored in a SQL Server Express database.

A copy of this database is included in the downloadable code from the Wrox Web site (`www.wrox.com`*).*

After the design surface is open and ready, open the Visual Studio Server Explorer tool and locate the Movies database and expand the database's Tables folder. Drag the Movies table from the Server Explorer onto the design surface. Notice that as soon as you drop the database table onto the design surface, it is automatically interrogated to identify its structure. A corresponding entity class is created by the designer and shown on the design surface.


Add New Item - C:\Users\devinr\Documents\Visual Studio Codename Orcas\Chapter9VB\
Templates:
Visual Studio installed templates
Web Form
AJAX Client Behavior
AJAX Master Page
Browser File
DataSet
HTML Page
Report
Site Map
Style Sheet
Web Configuration File
XML Schema
Master Page
AJAX Client Control
AJAX Web Form
Class
Generic Handler
JScript File
Report Wizard
Skin File
Text File
Web Service
XSLT File
Web User Control
AJAX Client Library
AJAX-enabled WCF Service
Class Diagram
Global Application Class
LINQ to SQL Classes
Resource File
SQL Server Database
WCF Service
XML File
My Templates
Search Online Templates...
Add
Cancel

LINQ to SQL classes mapped to relational objects.

Name: Movies.dbml

Language: Visual C#

☐ Place code in separate file

☐ Select master page

Figure 9-8 The Add New Item dialog includes the new Linq to SQL File type

When you drop table objects onto the LINQ to SQL design surface, Visual Studio examines the entities name and will if necessary, attempt to automatically pluralize the class names it generated. It does this in order to help you more closely following the .NET Framework class naming standards. For example, if you drop the Products table from the Northwind database onto the design surface, it would automatically choose the singular name Product as the name of the generated class.

Unfortunately, while the designer generally does a pretty good job at figuring out the correct pluralization for the class names, it's not 100% accurate. Case in point, simply look at how it incorrectly pluralizes the Movies table to Movy when you drop it into the design surface. Thankfully the designer also allows you to change the name of entities on the design surface. You can do this simply by selecting the entity on the design surface and clicking on the entities name in designer.

Once you have added the Movie entity, drag the Genres table onto the design surface. Again, Visual Studio creates a class representation of this table (and notice it gives it the singular name Genre). Additionally, it detects an existing foreign key relationship between the Movie and Genre. Because it detects this relationship, a dashed line is added between the two tables. The lines arrow indicates the direction of the foreign key relationship that exists between the two tables. The LINQ to SQL design surface with Movies and Genres tables added is shown in Figure 9-9.

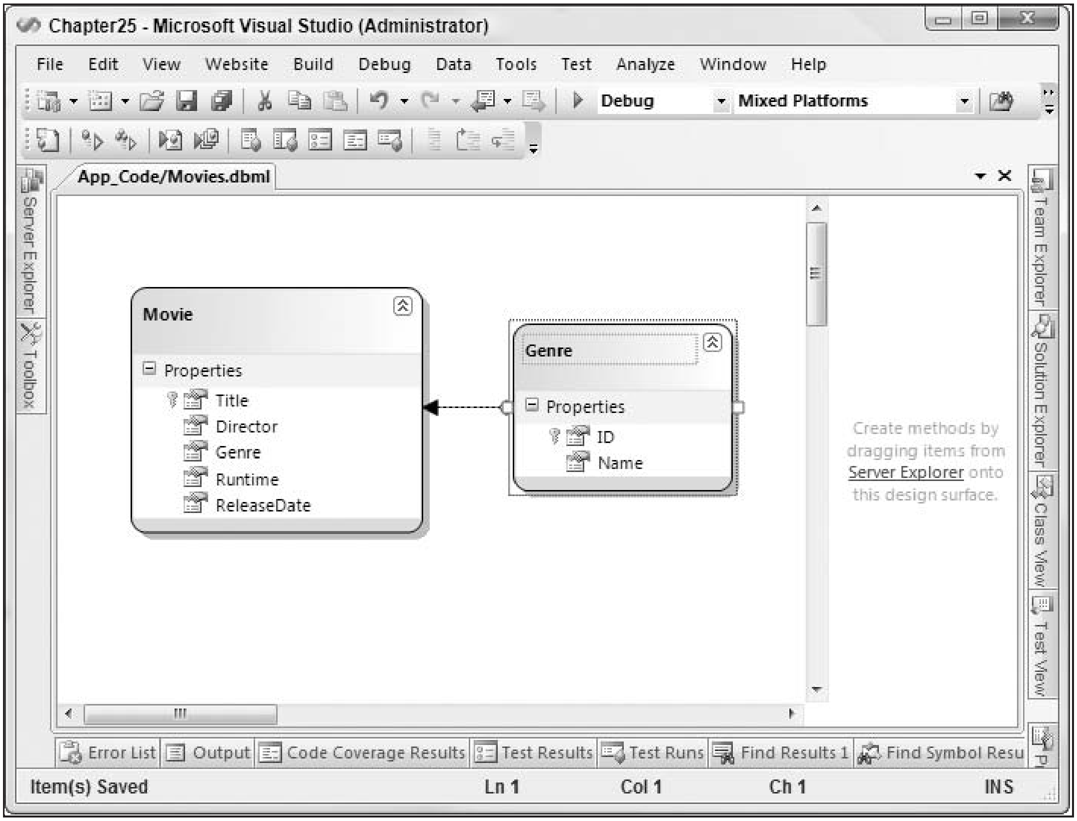

Chapter25 - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Data Tools Test Analyze Window Help
Debug
Mixed Platforms
App_Code/Movies.dbml
Server Explorer
Toolbox
Movie
Properties
Title
Director
Genre
Runtime
ReleaseDate
Genre
Properties
ID
Name
Create methods by dragging items from Server Explorer onto this design surface.
Team Explorer
Solution Explorer
Class View
Test View
Error List
Output
Code Coverage Results
Test Results
Test Runs
Find Results 1
Find Symbol Resu
Item(s) Saved
Ln 1
Col 1
Ch 1
INS

Figure 9-9 The Movies and Genres tables after being added to the LINQ to SQL design surface

Now that you have set up your LINQ to SQL File, accessing its data context and querying its data is simple. To start, you need to create an instance of the data context in the Web page where you will be accessing the data, as shown in Listing 9-23.

Listing 9-23: Creating a new Data Context

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim dc As New MoviesDataContext()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
```

Continued

```
<head runat="server">
    <title> My Favorite Movies </title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:GridView ID="GridView1" runat="server">
        </asp:GridView>

    </div>
    </form>
</body>
</html>
```

C#

```
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        MoviesDataContext dc = new MoviesDataContext();

    }
</script>
```

In this case you created an instance of the MoviesDataContext, which is the name of the data context class generated by the LINQ to SQL File you added earlier.

Because the data context class is automatically generated by the LINQ to SQL file, its name will change each time you create a new LINQ to SQL file. The name of this class is determined by appending the name of your LINQ to SQL Class file with the DataContext suffix, so had you named your LINQ to SQL File Northwind.dbml, the data context class would have been NorthwindDataContext.

After you have added the data context to your page, you can begin writing LINQ queries against it. As mentioned earlier, because LINQ to SQL generated object classes mirror the structure of our database tables, you will get Intellisense support as you write your LINQ queries. Listing 9-24 shows the same basic Movie listing query that has been shown in prior sections.

Listing 9-24: Querying Movie data from LINQ to SQL

VB

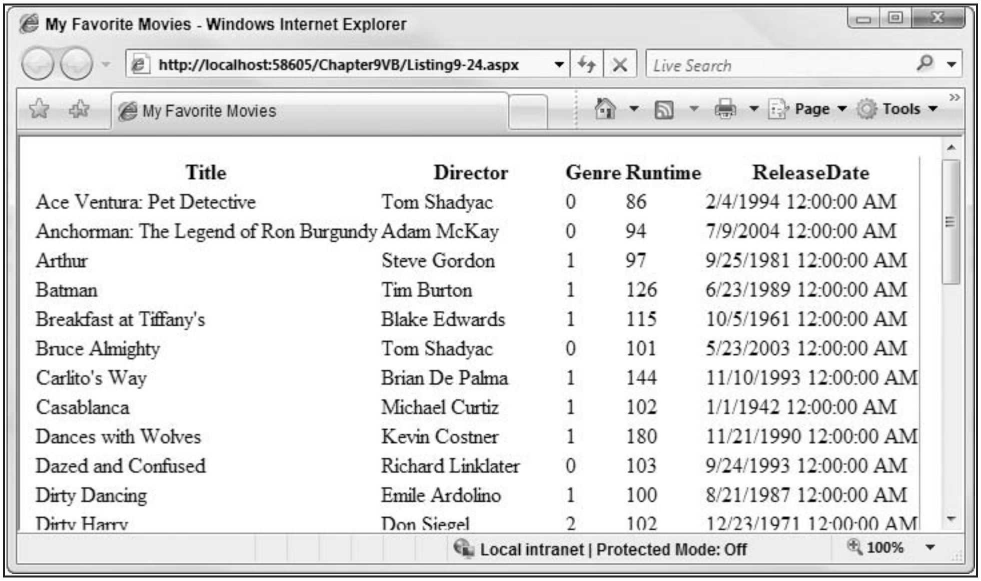
```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim dc As New MoviesDataContext()

    Dim query = From m In dc.Movies _
                Select m

    Me.GridView1.DataSource = query
    Me.GridView1.DataBind()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    MoviesDataContext dc = new MoviesDataContext();
```

```
        var query = from m in dc.Movies
                    select m;

        this.GridView1.DataSource = query;
        this.GridView1.DataBind();
    }
```

As is shown in Figure 9-10, running the code generates a raw list of the Movies in our database.


My Favorite Movies - Windows Internet Explorer
http://localhost:58605/Chapter9VB/Listing9-24.aspx
Live Search
My Favorite Movies
Page
Tools
Local intranet | Protected Mode: Off
100%

Title	Director	Genre	Runtime	ReleaseDate
Ace Ventura: Pet Detective	Tom Shadyac	0	86	2/4/1994 12:00:00 AM
Anchorman: The Legend of Ron Burgundy	Adam McKay	0	94	7/9/2004 12:00:00 AM
Arthur	Steve Gordon	1	97	9/25/1981 12:00:00 AM
Batman	Tim Burton	1	126	6/23/1989 12:00:00 AM
Breakfast at Tiffany's	Blake Edwards	1	115	10/5/1961 12:00:00 AM
Bruce Almighty	Tom Shadyac	0	101	5/23/2003 12:00:00 AM
Carlito's Way	Brian De Palma	1	144	11/10/1993 12:00:00 AM
Casablanca	Michael Curtiz	1	102	1/1/1942 12:00:00 AM
Dances with Wolves	Kevin Costner	1	180	11/21/1990 12:00:00 AM
Dazed and Confused	Richard Linklater	0	103	9/24/1993 12:00:00 AM
Dirty Dancing	Emile Ardolino	1	100	8/21/1987 12:00:00 AM
Dirty Harry	Don Siegel	2	102	12/23/1971 12:00:00 AM

Figure 9-10 Data retrieved from a basic LINQ to SQL query

Note that we did not have to write any of the database access code that would typically have been required to create this page. LINQ has taken care of that for us, even generating the SQL query based on our LINQ syntax. You can see the SQL that LINQ generated for the query by writing the query to the Visual Studio output window, as shown in Listing 9-25.

Listing 9-25: Writing the LINQ to SQL query to the output window

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim dc As New MoviesDataContext()

    Dim query = From m In dc.Movies _
                Select m

    System.Diagnostics.Debug.WriteLine(query)

    Me.GridView1.DataSource = query
```

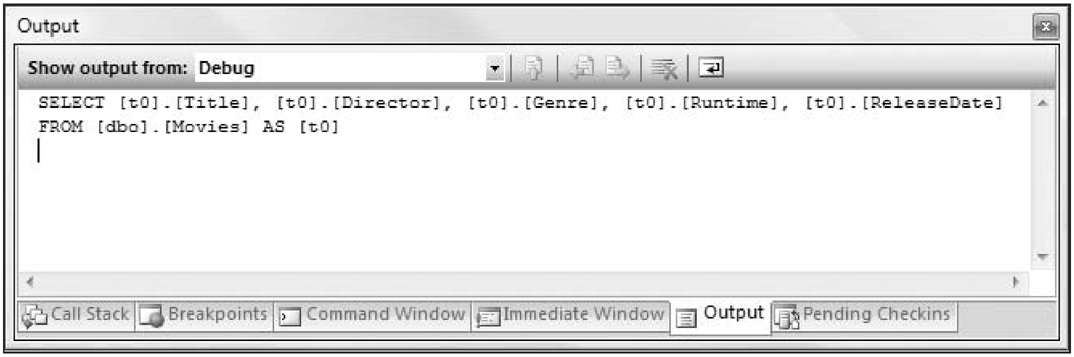
Continued

```
    Me.GridView1.DataBind()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    MoviesDataContext dc = new MoviesDataContext();

    var query = from m in dc.Movies
                select m;

    System.Diagnostics.Debug.WriteLine(query);

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

Now, when you debug the Web site using Visual Studio, you can see the SQL query, as shown in Figure 9-11.


Output
Show output from: Debug
Call Stack
Breakpoints
Command Window
Immediate Window
Output
Pending Checkins

```
SELECT [t0].[Title], [t0].[Director], [t0].[Genre], [t0].[Runtime], [t0].[ReleaseDate]
FROM [dbo].[Movies] AS [t0]
```

Figure 9-11 The SQL query generated by LINQ to SQL

As you can see, the SQL generated is standard SQL syntax, and LINQ is quite good at optimizing the queries it generates, even for more complex queries such as the grouping query shown in Listing 9-26.

Listing 9-26: Grouping LINQ to SQL data

VB

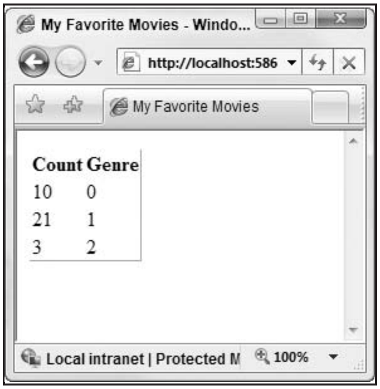
```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim dc As New MoviesDataContext()

    Dim query = From m In dc.Movies _
                Group By m.Genre Into g = Group, Count()

    System.Diagnostics.Debug.WriteLine(query)

    Me.GridView1.DataSource = query
```

```
        Me.GridView1.DataBind()

    End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    MoviesDataContext dc = new MoviesDataContext();

    var query = from m in dc.Movies
                group m by m.Genre into g
                select new { Genre  = g.Key, Count = g.Count() };

    System.Diagnostics.Debug.WriteLine(query);

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

The generated SQL for this query is shown in Figure 9-12.



Count	Genre
10	0
21	1
3	2

Figure 9-12 Grouping SQL data using LINQ to SQL

Note that SQL to LINQ generates SQL that is optimized for SQL Server 2005.

LINQ also includes a logging option you can enable by setting the Log property of the data context.

While LINQ to SQL does an excellent job generating the SQL query syntax, there may be times where it is more appropriate to use other SQL query methods, such as Stored Procedures, or Views. LINQ supports using the predefined queries as well.

To use a SQL View with LINQ to SQL, you simply drag the View onto the LINQ to SQL design surface just as you would a standard SQL table. Views appear on the design surface, just as the tables we added earlier. Once the View is on the design surface, you can execute queries against it, just as you did the SQL tables. This is shown in Listing 9-27.

Listing 9-27: Querying LINQ to SQL data using a View

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim dc As New MoviesDataContext()

    Dim query = From m In dc.AllMovies _
                Select m

    System.Diagnostics.Debug.WriteLine(query)

    Me.GridView1.DataSource = query
    Me.GridView1.DataBind()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    MoviesDataContext dc = new MoviesDataContext();

    var query = from m in dc.AllMovies
                select m;

    System.Diagnostics.Debug.WriteLine(query);

    this.GridView1.DataSource = query;
    this.GridView1.DataBind();
}
```

Unlike Tables or Views, which LINQ to SQL exposes as properties, Stored Procedures can require parameters. Therefore, LINQ to SQL exposes from the data context object them as method calls, allowing you to provide method parameter values which are translated by LINQ into store procedure parameters. Listing 2-28 shows a simple stored procedure you can use to retrieve a specific Genre from the database.

Listing 9-28: Simple SQL Stored procedure

```
CREATE PROCEDURE dbo.GetGenre
    (
    @id int
    )

AS
    SELECT * FROM Genre WHERE ID = @id
```

You can add a Stored Procedure to your LINQ to SQL designer just as you did the Tables and Views, by dragging them from the Server Explorer onto the LINQ to SQL Classes design surface. If you expect your stored procedure to return a collection of data from a table in your database, you should drop the stored procedure onto the LINQ class that represents the types returned by the query. In the case of the stored procedure shown in Listing 9-28, it will return all of the Genre records that match the provided ID, therefore you should drop the GetGenres stored procedure onto the Genres table in the Visual Studio designer. This tells the designer to generate a method which returns a generic collection of Genre objects. Once you drop the stored procedure onto the design surface, unlike the Tables and Views, the Stored Procedure will be displayed in a list on the right-hand side of the design surface. The GetGenre stored procedure is shown after being added in Figure 9-13.

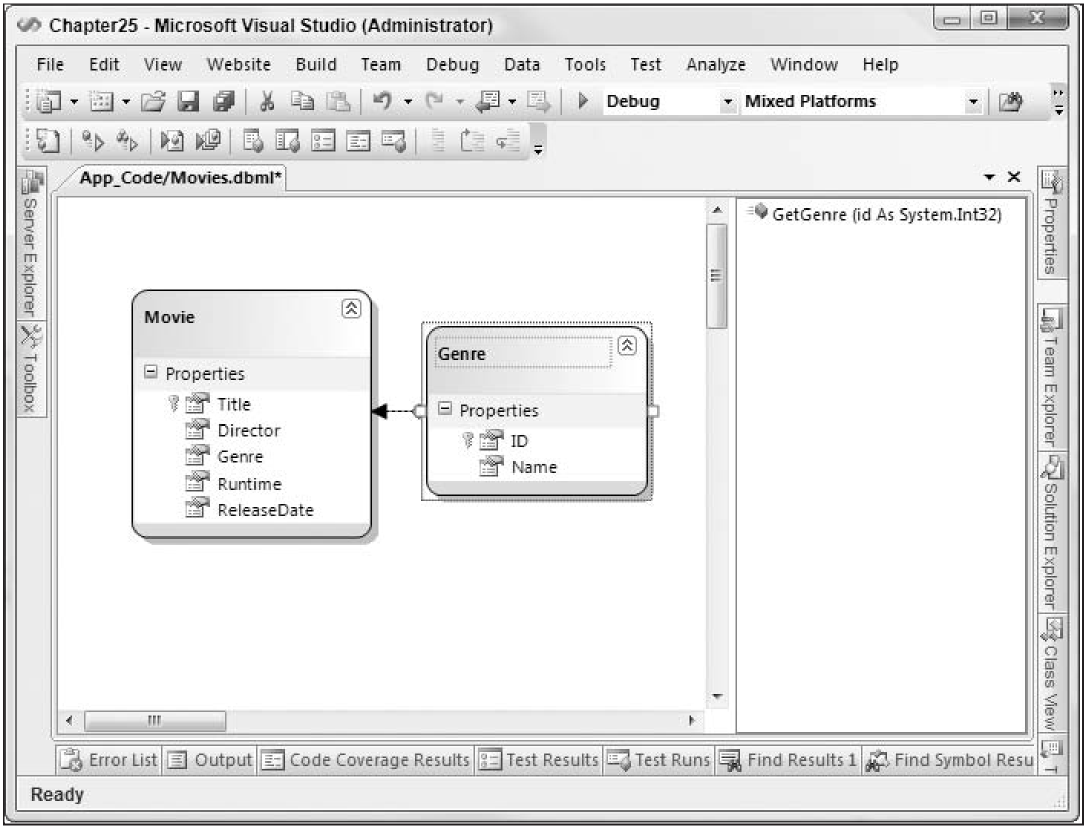

Chapter25 - Microsoft Visual Studio (Administrator)
File
Edit
View
Website
Build
Team
Debug
Data
Tools
Test
Analyze
Window
Help
Debug
Mixed Platforms
App_Code/Movies.dbml*
Server Explorer
Toolbox
Movie
Properties
Title
Director
Genre
Runtime
ReleaseDate
Genre
Properties
ID
Name
GetGenre (id As System.Int32)
Properties
Team Explorer
Solution Explorer
Class View
Error List
Output
Code Coverage Results
Test Results
Test Runs
Find Results 1
Find Symbol Resu
Ready

Figure 9-13 The GetGenre stored procedure has been added to the Method Pane in the LINQ to SQL design surface

After you have added the Stored Proceedures, you can access them through the data context, just as the Table and Views we accessed. As was stated earlier, however, LINQ to SQL exposes them as method calls. Therefore, they may require you to provide method parameters, as shown in Listing 9-29.

Listing 9-29: Selecting data from a Stored Procedure

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim dc As New MoviesDataContext()

    Me.GridView1.DataSource = dc.GetGenre(1)
    Me.GridView1.DataBind()
End Sub
```

Continued

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    MoviesDataContext dc = new MoviesDataContext();

    this.GridView1.DataSource = dc.GetGenre(1);
    this.GridView1.DataBind();
}
```

Insert, Update, and Delete Queries through LINQ

Not only can LINQ to SQL be used to create powerful queries that select data from a data source, but it can also manage insert, updates, and delete operations. By default, LINQ to SQL does this in much the same manner as selecting data. LINQ to SQL uses the object class representations of the SQL structures and dynamically generates SQL Insert, Update, and Delete commands. As with selection, you can also use Stored Procedures to perform the insert, update, or deletes.

Insert Data Using LINQ

Inserting data using LINQ to SQL is as easy as creating a new instance of the object you want to insert, and adding that to the object collection. The LINQ classes provide two methods called InsertOnSubmit and InsertAllOnSubmit which make it simple to create and add any object to a LINQ collection. The InsertOnSubmit accepts a single entity as its method parameter, allowing you to insert a single entity, while the InsertAllOnSubmit method accepts a collection as its method parameter, allowing you to insert an entire collection of data in a single method call.

Once you have added your objects, LINQ to SQL does require the extra step of calling the Data Context objects SubmitChanges method. Calling this method tells LINQ to initiate the Insert action. Listing 9-30 shows an example of creating a new Movies object, and then Adding it to the Movies collection and calling SubmitChanges to persist the change back to the SQL database.

Listing 9-30: Inserting data using LINQ to SQL

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim dc As New MoviesDataContext()

    Dim m As New Movie With {.Title = "The Princess Bride", .Director =
"Rob Reiner", _
        .Genre = 0, .ReleaseDate = DateTime.Parse("9/25/1987"), .Runtime = 98}

    dc.Movies.InsertOnSubmit(m)
    dc.SubmitChanges()

End Sub
```

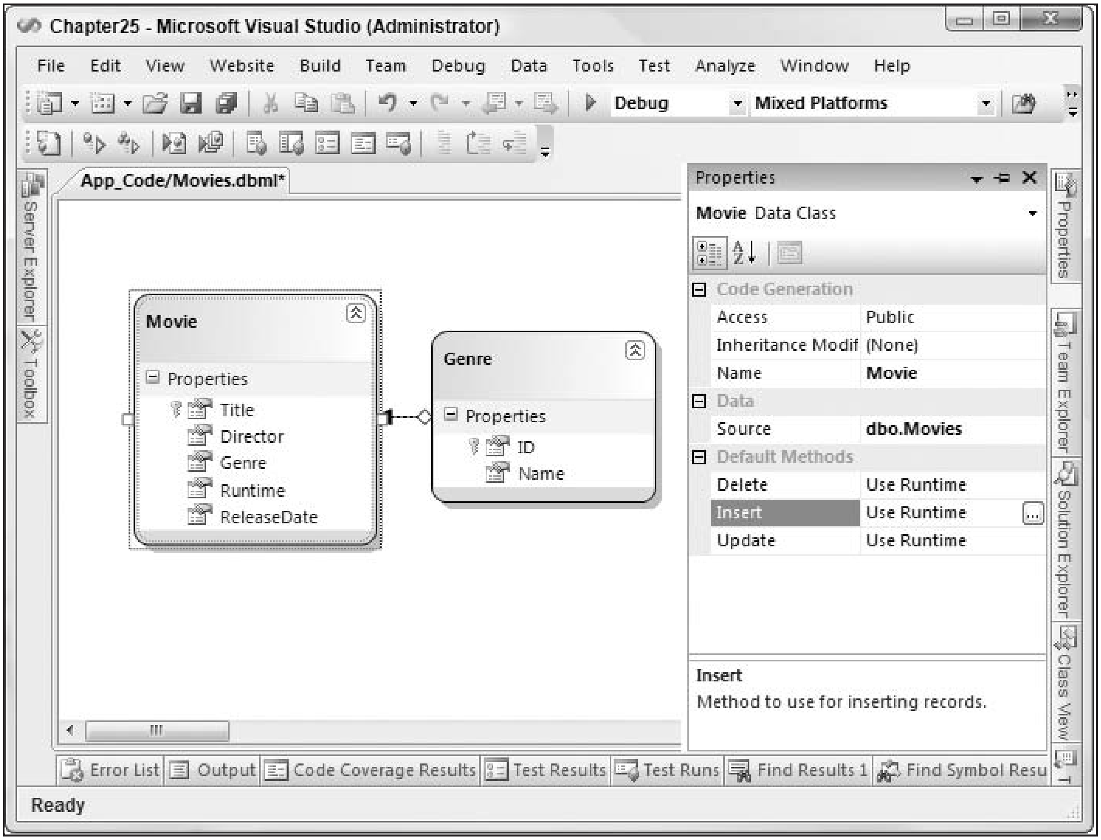
C#

```
protected void Page_Load(object sender, EventArgs e)
```

```
{
    MoviesDataContext dc = new MoviesDataContext();

    Movie m = new Movie { Title="The Princess Bride", Director="Rob Reiner", Genre=0,
        ReleaseDate=DateTime.Parse("9/25/1987"), Runtime=98 } ;

    dc.Movies.InsertOnSubmit(m);
    dc.SubmitChanges();
}
```

Using Stored Procedures to Insert Data

Of course, you may already have a complex stored procedure written to handle the insertion of data into your database table. LINQ makes it simple to use an existing Stored Procedure to insert data into a table. To do this, on the LINQ to SQL design surface, select the entity you want to insert data into, which in this case is the Movies entity. After selecting the entity, open its properties window and locate the Default Methods section, as shown in Figure 9-14.


Chapter25 - Microsoft Visual Studio (Administrator)
File Edit View Website Build Team Debug Data Tools Test Analyze Window Help
Debug
Mixed Platforms
App_Code/Movies.dbml*
Server Explorer
Toolbox
Movie
Properties
Title
Director
Genre
Runtime
ReleaseDate
Genre
Properties
ID
Name
Properties
Movie Data Class
Team Explorer
Solution Explorer
Class View
Error List
Output
Code Coverage Results
Test Results
Test Runs
Find Results 1
Find Symbol Resu
Ready

Code Generation
Access: Public
Inheritance Modif: (None)
Name: Movie
Data
Source: dbo.Movies
Default Methods
Delete: Use Runtime
Insert: Use Runtime
Update: Use Runtime

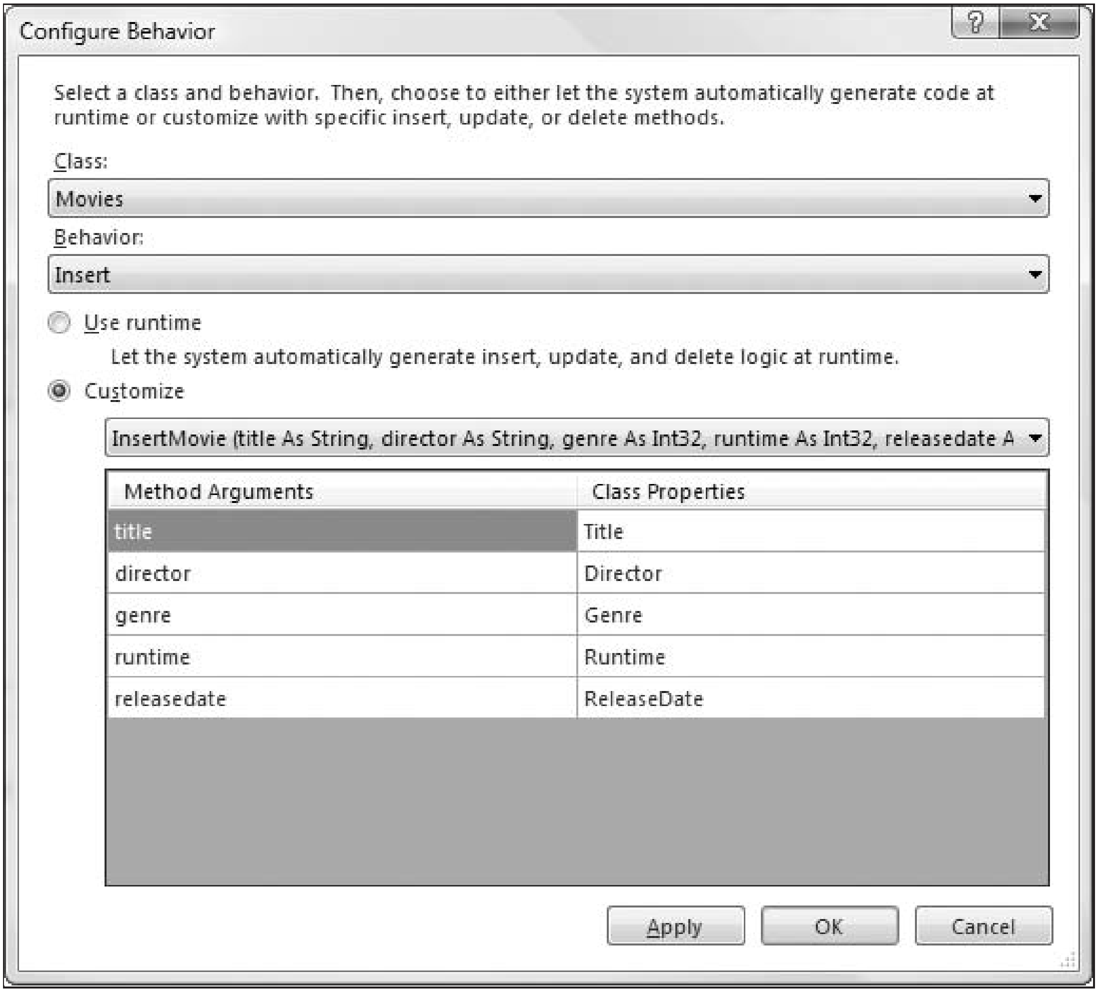
Insert
Method to use for inserting records.

Figure 9-14 The properties of the Movies Default Methods section enable you to configure the Delete, Insert, and Update behavior of LINQ to SQL for this table

The Default Methods section contains three properties, Delete, Insert, and Update, which define the behavior LINQ should use when executing these actions on the Movies table. By default, each property is set to the value `UseRuntime`, which tells LINQ to dynamically generate SQL statements at runtime. Because you want to insert data into the table using a Stored Procedure, open the Insert properties Configure Behavior dialog.

In the dialog, change the Behavior radio button selection from Use Runtime to Customize. Next, select the appropriate stored procedure from the drop-down list below the radio buttons. When you select the stored procedure, LINQ automatically tries to match the table columns to the stored procedure input parameters. However, you can change these manually, if needed.

The final Configure Behavior dialog is shown in Figure 9-15.



Configure Behavior

Select a class and behavior. Then, choose to either let the system automatically generate code at runtime or customize with specific insert, update, or delete methods.

Class: Movies

Behavior: Insert

○ Use runtime

Let the system automatically generate insert, update, and delete logic at runtime.

● Customize

InsertMovie (title As String, director As String, genre As Int32, runtime As Int32, releasedate A

Method Arguments	Class Properties
title	Title
director	Director
genre	Genre
runtime	Runtime
releasedate	ReleaseDate

Figure 9-15 Configure Behavior dialog configured for the InsertMovie Stored Procedure

Now, when you run the code from Listing 9-30, LINQ will use the stored procedure you configured instead of dynamically generating a SQL Insert statement.

Update Data using LINQ

Updating data with LINQ is very similar to inserting data. The first set is to get the specific object you want to update. You can do this by using the Single method of the collection you want to change. The scalar Single method returns a single object from the collection based on its input parameter. If more than one record matches the parameters, the Single method simply returns the first match.

Once you have the record you want to update, you simply change the object's property values, and then call the data context's SubmitChanges method. Listing 9-31 shows the code required to update a specific Movie.

Listing 9-31: Updating data using LINQ to SQL

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim dc As New MoviesDataContext()

    Dim movie = dc.Movies.Single(Function(m) m.Title = "Fletch")
    movie.Genre = 1

    dc.SubmitChanges()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    MoviesDataContext dc = new MoviesDataContext();

    var movie = dc.Movies.Single(m => m.Title == "Fletch");
    movie.Genre = 1;

    dc.SubmitChanges();
}
```

Handling Data Concurrency

LINQ to SQL also includes and uses by default optimistic concurrency. That means that if two users retrieve the same record from the database and both try to update it, the first user to submit their update to the server wins. If the second user attempts to update the record after the first, LINQ to SQL will detect that the original record has changed and will raise a ChangeConflictException.

Deleting Data Using LINQ

Finally, LINQ to SQL also enables you to delete data from your SQL data source. Each data class object generated by the LINQ to SQL designer also includes two methods that enable you to delete objects from the collection, the `DeleteOnSubmit` and `DeleteAllOnSubmit` methods. As the names imply, the `DeleteOnSubmit` method removes a single object from the collection, whereas the `DeleteAllOnSubmit` method removes all records from the collection.

Listing 9-32 shows how you can use LINQ and the `DeleteOnSubmit` and `DeleteAllOnSubmit` methods to delete data from your data source.

Listing 9-32: Deleting data using LINQ to SQL

VB

```
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim dc As New MoviesDataContext()

        'Select and remove all Action movies
        Dim query = From m In dc.Movies _
                    Where (m.Genre = 2) _
                    Select m

        dc.Movies.DeleteAllOnSubmit(query)

        'Select a single movie and remove it
        Dim movie = dc.Movies.Single(Function(m) m.Title = "Fletch")
        dc.Movies.DeleteOnSubmit(movie)

        dc.SubmitChanges()
    End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    MoviesDataContext dc = new MoviesDataContext();

    //Select and remove all Action movies
    var query = from m in dc.Movies
                where m.Genre == 2
                select m;

    dc.Movies.DeleteAllOnSubmit(query);

    //Select a single movie and remove it
    var movie = dc.Movies.Single(m => m.Title == "Fletch");
    dc.Movies.DeleteOnSubmit(movie);

    dc.SubmitChanges();
}
```

As with the other SQL commands, you must remember to call the Data Contexts `SubmitChanges` method in order to commit the changes back to the SQL data source.

Extending LINQ

This chapter focuses primarily on the LINQ capabilities included in the .NET framework, but LINQ is highly extensible and can be used to create query frameworks over just about any data source. While showing you how to implement you own LINQ provider is beyond the scope of this chapter, there are lots of implementations of LINQ that query a wide variety of data stores such as LDAP, Sharepoint, and even `Amazon.com`.

Roger Jennings from Oakleaf Systems maintains a list of third-party LINQ providers on his blog which can be found at `http://oakleafblog.blogspot.com/2007/03/third-party-linq-providers.html`.

Summary

This chapter has introduced you to the new Language Integrated Query, or LINQ, features of .NET 3.5, which greatly simplifies the querying of data in .NET. LINQ makes query a first class concept, embedded directly in .NET.

In this review of LINQ, we first looked at the current methods for performing object queries, including basic data filtering, grouping, and sorting. We looked at the shortcomings of current object query techniques, including the requirement for developers to not only define what the query should do, but also exactly how it should do it. Additionally, we looked at how even simple operations can result in highly complex code that can be difficult to read and maintain.

Next, we began looking at the three basic types of LINQ — LINQ to Objects, LINQ to XML, and LINQ to SQL. Each flavor of LINQ uses the same basic query syntax to dramatically simplify the querying of Objects, XML or SQL. We looked at how you can use the basic SQL-like query syntax for selection, filtering, and grouping. This query syntax is clean and easily readable, and also includes many of the same operator features as SQL.

We also looked at the basic O/R mapper that is included with LINQ to SQL. The O/R mapper makes it easy to create CLR objects that represent SQL structures such as Tables, Views, and Stored Procedures. Once the CLR objects are created, LINQ can be used to query the objects.

Finally we looked at how using LINQ to SQL, you can easily change the data in your database, using generated SQL statements, or using custom Stored Procedures.

10

Working with XML and LINQ to XML

This is not a book about XML, the eXtensible Markup Language; but XML has become such a part of an ASP.NET programmer's life that the topic deserves its own chapter. Although most of the XML functionality in the .NET Framework appears to be in the `System.Xml` namespace, you can find XML's influence throughout the entire Framework including `System.Data` and `System.Web`.

XML is oft maligned and misunderstood. To some, XML is simply a text-based markup language; to others it is an object serialization format or a document-encoding standard. In fact, XML has become the de facto standard manner in which data passes around the Internet. XML, however, is not really a technology as much as it is a set of standards or guiding principles. It provides a structure within which data can be stored; but the XML specification doesn't dictate how XML processors, parsers, formatters, and data access methods should be written or implemented. `System.Xml`, `System.Xml.Linq`, and other namespaces contain the .NET Framework 3.5's view on how programmers should manipulate XML. Some of its techniques, such as XSLT and XML Schema, are standards-based. Others, like `XmlReader` and `XmlWriter`, started in the world of the .NET Framework and now Java has similar classes. The .NET Framework 3.5 — along with new compilers for C# 3.0 and VB 9 — brings LINQ and LINQ to XML as a Language-Integrated Query over XML to the table.

This is an ASP.NET book, aimed at the professional Web developer, so it can't be a book all about LINQ. However, a single chapter can't do LINQ justice. Rather than making this a chapter that focuses exclusively on just `System.Xml` or `System.Xml.Linq`, this chapter will present the new LINQ model and syntax as a juxtaposition to the way you're used to manipulating XML. The examples will include both the traditional and the new LINQ way of doing things. We recognize that you won't go and rewrite all your `System.Xml` code to use LINQ just because it's cool, but seeing the new syntax alongside what you are used to is an excellent way to learn the syntax, and it also assists you in making decisions on which technology to use going forward.

In this chapter some listings will include a 'q' in the numbering scheme. These listings demonstrate how you can use LINQ to XML to accomplish the same task shown in the previous related listing. For example, Listing 10-5q shows the way you'd accomplish the task from Listing 10-5 using LINQ to XML.

You'll learn more about LINQ and its flexibility in the all new chapter dedicated to the technology. For the purposes of this chapter, know that `System.Xml.Linq` introduces an all new series of objects such as XDocument and XElement that in some ways complement the existing APIs, but in many ways, eclipse them. You'll also see how these new classes have provided ''bridges'' back and forth between `System.Xml` and `System.Xml.Linq` that will enable you to use many new techniques for clearer, simpler code, while still utilizing the very useful, powerful (and well-tested) features of the `System.Xml` classes you're used to.

Ultimately, however, remember that while the .NET Framework has its own unique style of API around the uses of XML, the XML consumed and produced by these techniques is standards-based and can be used by other languages that consume XML. This chapter covers all the major techniques for manipulating XML provided by the .NET Framework. `XmlReader` and `XmlWriter` offer incredible speed but may require a bit more thought. The XmlDocument or DOM is the most commonly used method for manipulating XML but you'll pay dearly in performance penalties without careful use. ADO .NET DataSets have always provided XML support, and their deep support continues with .NET 3.5. XML Stylesheet Tree Transformations (XSLT) gained debugging capabilities in Visual Studio 2005, and is improved with new features in Visual Studio 2008, such as XSLT Data Breakpoints and better support in the editor for loading large documents. Additionally, XSLT stylesheets can be compiled into assemblies even more easily with the new command-line stylesheet compiler. ASP.NET continues to make development easier with some simple yet powerful server controls to manipulate XML.

Its flexibility and room for innovation make XML very powerful and a joy to work with.

Note that when the acronym XML appears by itself, the whole acronym is capitalized, but when it appears in a function name or namespace, only the X is capitalized, as in `System.Xml` or `XmlTextReader`. Microsoft's API Design Guidelines dictate that if an abbreviation of three or more characters appears in a variable name, class name, or namespace, the first character is capitalized.

The Basics of XML

Listing 10-1, a `Books.xml` document that represents a bookstore's inventory database, is one of the sample documents used in this chapter. This example document has been used in various MSDN examples for many years.

Listing 10-1: The Books.xml XML document

```
<?xml version='1.0'?>
<!-- This file is a part of a book store inventory database -->
<bookstore xmlns="http://example.books.com">
```

```
        <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
            <title>The Autobiography of Benjamin Franklin</title>
            <author>
                <first-name>Benjamin</first-name>
                <last-name>Franklin</last-name>
            </author>
            <price>8.99</price>
        </book>
        <book genre="novel" publicationdate="1967" ISBN="0-201-63361-2">
            <title>The Confidence Man</title>
            <author>
                <first-name>Herman</first-name>
                <last-name>Melville</last-name>
            </author>
            <price>11.99</price>
        </book>
        <book genre="philosophy" publicationdate="1991" ISBN="1-861001-57-6">
            <title>The Gorgias</title>
            <author>
                <first-name>Sidas</first-name>
                <last-name>Plato</last-name>
            </author>
            <price>9.99</price>
        </book>
    </bookstore>
```

The first line of Listing 10-1, starting with `<?xml version='1.0'?>`, is an XML declaration. This line should always appear before the first element in the XML document and indicates the version of XML with which this document is compliant.

The second line is an XML comment and uses the same syntax as an HTML comment. This isn't a coincidence; remember that XML and HTML are both descendants of SGML, the Standard Generalized Markup Language. Comments are always optional in XML documents

The third line, `<bookstore>`, is the opening tag of the root element or document entity of the XML document. An XML document can have only one root element. The last line in the document is the closing tag `</bookstore>` of the root element. No elements of the document can appear after the final closing tag `</bookstore>`. The `<bookstore>` element contains an `xmlns` attribute such as `xmlns="http://example.books.com"`. Namespaces in XML are similar to namespaces in the .NET Framework because they provide *qualification of elements and attributes*. It's very likely that someone else in the world has created a bookstore XML document before, and it's also likely he or she chose an element such as `<book>` or `<bookstore/>`. A namespace is defined to make your `<book>` element different from any others and to deal with the chance that other `<book>` elements might appear with yours in the same document — it's possible with XML.

This namespace is often a URL (Uniform/Universal Resource Locator), but it actually can be a URI (Uniform/Universal Resource Identifier). A namespace can be a GUID or a nonsense string such as `www-computerzen-com:schema` as long as it is unique. Recently, the convention has been to use a URL because URLs are ostensibly unique, thus making the document's associated schema unique. You learn more about schemas and namespaces in the next section.

The fourth line is a little different because the `<book>` element contains some additional attributes such as `genre`, `publicationdate`, and `ISBN`. The order of the elements matters in an XML document, but the order of the attributes does not. These attributes are said to be *on* or *contained within* the book element. Consider the following line of code:

```
<book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
```

Notice that every element following this line has a matching end tag, similar to the example that follows:

```
<example>This is a test</example>
```

If no matching end tag is used, the XML is not well formed; technically it isn't even XML! These next two example XML fragments are not well formed because the elements don't match up:

```
<example>This is a test

<example>This is a test</anothertag>
```

If the `<example>` element is empty, it might appear like this:

```
<example></example>
```

Alternatively, it could appear as a shortcut like this:

```
<example/>
```

The syntax is different, but the semantics are the same. The difference between the syntax and the semantics of an XML document is crucial for understanding what XML is trying to accomplish. XML documents are text files by their nature, but the information — the information set — is representable using text that isn't exact. The set of information is the same, but the actual bytes are not.

> **Note that attributes appear only within start tags or empty elements such as `<book genre="scifi"></book>` or `<book genre = "scifi" />`.Visit the World Wide Web Consortium's (W3C) XML site at `www.w3.org/XML/` for more detailed information on XML.**

The XML InfoSet

The XML InfoSet is a W3C concept that describes what is and isn't significant in an XML document. The InfoSet isn't a class, a function, a namespace, or a language — the InfoSet is a concept.

Listing 10-2 describes two XML documents that are syntactically different but semantically the same.

Listing 10-2: XML syntax versus semantics

XML document

```
<?xml version='1.0'?>
<bookstore>
    <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
        <title>The Autobiography of Benjamin Franklin</title>
        <author>
```

```
            <first-name>Benjamin</first-name>
            <last-name>Franklin</last-name>
        </author>
        <price></price>
    </book>
</bookstore>
```

XML document that differs in syntax, but not in semantics

```
<?xml version='1.0'?><bookstore><book genre="autobiography"
publicationdate="1981" ISBN="1-861003-11-0"><title>The Autobiography of Benjamin
Franklin</title><author><first-name>Benjamin</first-name>
<last-name>Franklin</last-name></author><price/></book></bookstore>
```

Certainly, the first document in Listing 10-2 is easier for a human to read, but the second document is just as easy for a computer to read. The second document has insignificant white space removed.

Notice also that the empty `<price/>` element is different in the two documents. The first uses the verbose form, whereas the second element uses the shortcut form to express an empty element. However, *both are empty elements*.

You can manipulate XML as elements and attributes. You can visualize XML as a tree of nodes. You rarely, if ever, have to worry about angle brackets or parse text yourself. A text-based differences (diff) tool would report these two documents are different because their character representations are different. An XML-based differences tool would report (correctly) that they are the same document. Each document contains the same InfoSet.

You can run a free XML Diff Tool online at `http://www.deltaxml.com/free/compare//`.

XSD–XML Schema Definition

XML documents must be well formed at the very least. However, just because a document is well formed doesn't ensure that its elements are in the right order, have the right name, or are the correct data types. After creating a well-formed XML document, you should ensure that your document is also *valid*. A *valid* XML document is well formed and also has an associated XML Schema Definition (XSD) that describes what elements, simple types, and complex types are allowed in the document.

The schema for the `Books.xml` file is a glossary or vocabulary for the bookstore described in an XML Schema definition. In programming terms, an XML Schema is a type definition, whereas an XML document is an instance of that type. Listing 10-3 describes one possible XML Schema called `Books.xsd` that validates against the `Books.xml` file.

Listing 10-3: The Books.xsd XML Schema

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://example.books.com"
xmlns="http://example.books.com"
targetNamespace="http://example.books.com"
elementFormDefault="qualified">
```

```
        <xsd:element name="bookstore" type="bookstoreType"/>

        <xsd:complexType name="bookstoreType">
            <xsd:sequence maxOccurs="unbounded">
                <xsd:element name="book" type="bookType"/>
            </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="bookType">
            <xsd:sequence>
                <xsd:element name="title" type="xsd:string"/>
                <xsd:element name="author" type="authorName"/>
                <xsd:element name="price" type="xsd:decimal"/>
            </xsd:sequence>
            <xsd:attribute name="genre" type="xsd:string"/>
            <xsd:attribute name="publicationdate" type="xsd:string"/>
            <xsd:attribute name="ISBN" type="xsd:string"/>
        </xsd:complexType>

        <xsd:complexType name="authorName">
            <xsd:sequence>
                <xsd:element name="first-name" type="xsd:string"/>
                <xsd:element name="last-name" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:schema>
```

The XML Schema in Listing 10-3 starts by including a series of namespace prefixes used in the schema document as attributes on the root element. The prefix `xsd:` is declared on the root element (`xmlns:xsd="http://www.w3.org/2001/XMLSchema"`) and then used on all other elements of that schema. The default namespace assumed for any elements without prefixes is described by the `xmlns` attribute like this:

```
xmlns="http://example.books.com"
```

A namespace-qualified element has a prefix such as `<xsd:element>`. The target namespace for all elements in this schema is declared with the `targetNamespace` attribute.

XML Schema can be daunting at first; but if you read each line to yourself as a *declaration*, it makes more sense. For example, the line

```
<xsd:element name="bookstore" type="bookstoreType"/>
```

declares that an element named `bookstore` has the type `bookstoreType`. Because the `targetNamespace` for the schema is `http://example.books.com`, that is the namespace of each declared type in the `Books.xsd` schema. If you refer to Listing 10-1, you see that the namespace of the `Books.xml` document is also `http://example.books.com`.

For more detailed information on XML Schema, visit the W3C's XML Schema site at `www.w3.org/XML/Schema`.

Editing XML and XML Schema in Visual Studio 2008

If you start up Visual Studio 2008 and open the `Books.xml` file into the editor, you notice immediately that the Visual Studio editor provides syntax highlighting and formats the XML document as a nicely indented tree. If you start writing a new XML element anywhere, you don't have access to IntelliSense. Even though the `http://example.books.com` namespace is the default namespace, Visual Studio 2008 has no way to find the `Books.xsd` file; it could be located anywhere. Remember that the namespace is *not* a URL. It's a URI — an identifier. Even if it were a URL it wouldn't be appropriate for the editor, or any program you write, to go out on the Web looking for a schema. You have to be explicit when associating XML Schema with instance documents.

Classes and methods are used to validate XML documents when you are working programmatically, but the Visual Studio editor needs a hint to find the `Book.xsd` schema. Assuming the `Books.xsd` file is in the same directory as `Books.xml`, you have three ways to inform the editor:

- ❑ Open the `Books.xsd` schema in Visual Studio in another window while the `Books.xml` file is also open.
- ❑ Include a `schemaLocation` attribute in the `Books.xml` file.
- ❑ If you open at least one XML file with the `schemaLocation` attribute set, Visual Studio uses that schema for any other open XML files that don't include the attribute.
- ❑ Add the `Books.xsd` schema to the list of schemas that Visual Studio knows about internally by adding it to the `Schemas` property in the document properties window of the `Books.xml` file. When schemas are added in this way, Visual Studio checks the document's namespace and determines if it already knows of a schema that matches.

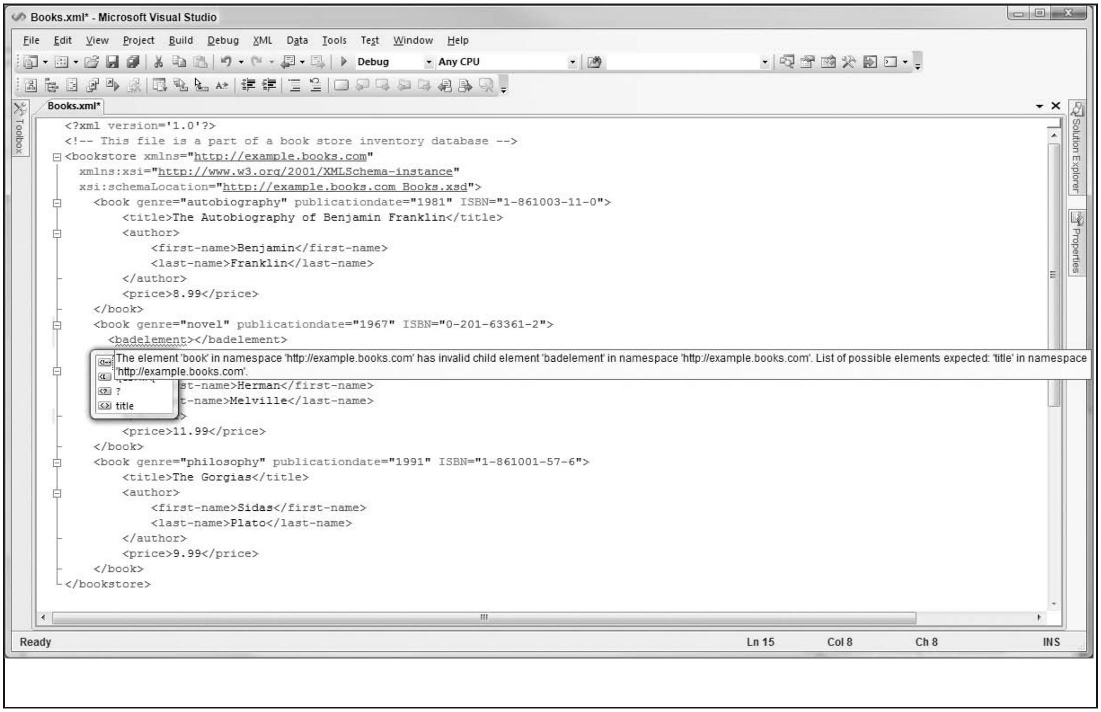
The `schemaLocation` attribute is in a different namespace, so include the `xmlns` namespace attribute and your chosen prefix for the schema's location, as shown in Listing 10-4.

Listing 10-4: Updating the Books.xml file with a schemaLocation attribute

```
<?xml version='1.0'?>
<!-- This file is a part of a book store inventory database -->
<bookstore xmlns="http://example.books.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://example.books.com Books.xsd">
    <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
        <title>The Autobiography of Benjamin Franklin</title>
        ...Rest of the XML document omitted for brevity...
```

The format for the `schemaLocation` attribute consists of pairs of strings separated by spaces where the first string in each pair is a namespace URI and the second string is the location of the schema. The location can be relative, as shown in Listing 10-4, or it can be an `http://` URL or `file://` location.

When the `Books.xsd` schema can be located for the `Books.xml` document, Visual Studio 2008's XML Editor becomes considerably more useful. Not only does the editor underline incorrect elements with blue squiggles, it also includes tooltips and IntelliSense for the entire document, as shown in Figure 10-1.


Books.xml* - Microsoft Visual Studio
File Edit View Project Build Debug XML Data Tools Test Window Help
Debug
Any CPU
Books.xml*
Toolbox
Solution Explorer
Properties
?
title
Ready
Ln 15
Col 8
Ch 8
INS

```
<?xml version='1.0'?>
<!-- This file is a part of a book store inventory database -->
<bookstore xmlns="http://example.books.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://example.books.com Books.xsd">
    <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
        <title>The Autobiography of Benjamin Franklin</title>
        <author>
            <first-name>Benjamin</first-name>
            <last-name>Franklin</last-name>
        </author>
        <price>8.99</price>
    </book>
    <book genre="novel" publicationdate="1967" ISBN="0-201-63361-2">
      <badelement></badelement>

                st-name>Herman</first-name>
                t-name>Melville</last-name>
                >
        <price>11.99</price>
    </book>
    <book genre="philosophy" publicationdate="1991" ISBN="1-861001-57-6">
        <title>The Gorgias</title>
        <author>
            <first-name>Sidas</first-name>
            <last-name>Plato</last-name>
        </author>
        <price>9.99</price>
    </book>
</bookstore>
```

The element 'book' in namespace 'http://example.books.com' has invalid child element 'badelement' in namespace 'http://example.books.com'. List of possible elements expected: 'title' in namespace 'http://example.books.com'.

Figure 10-1

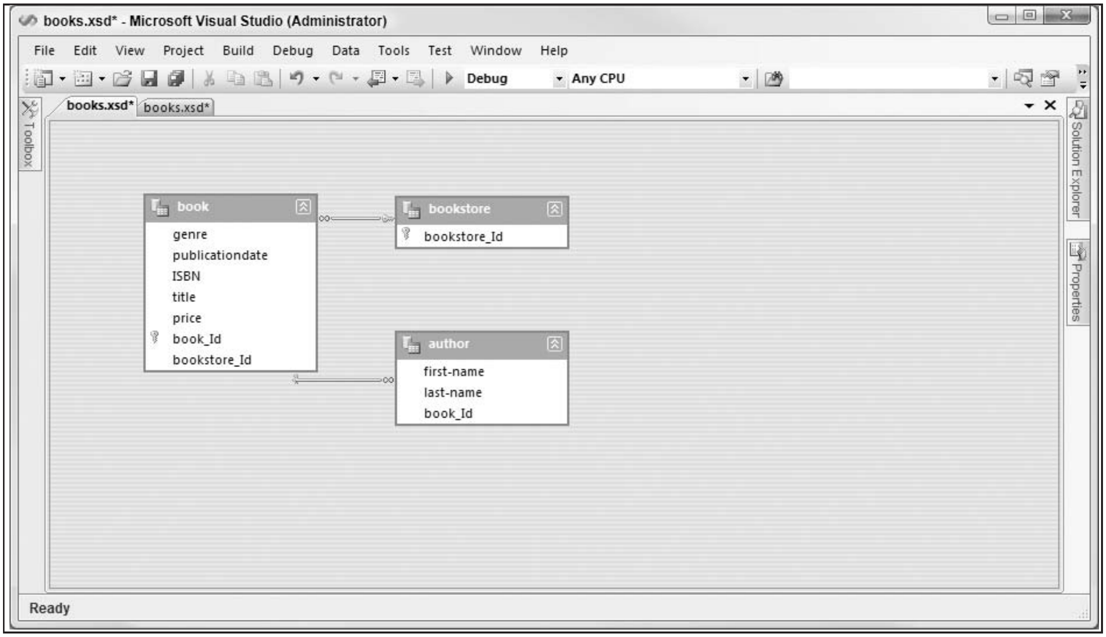
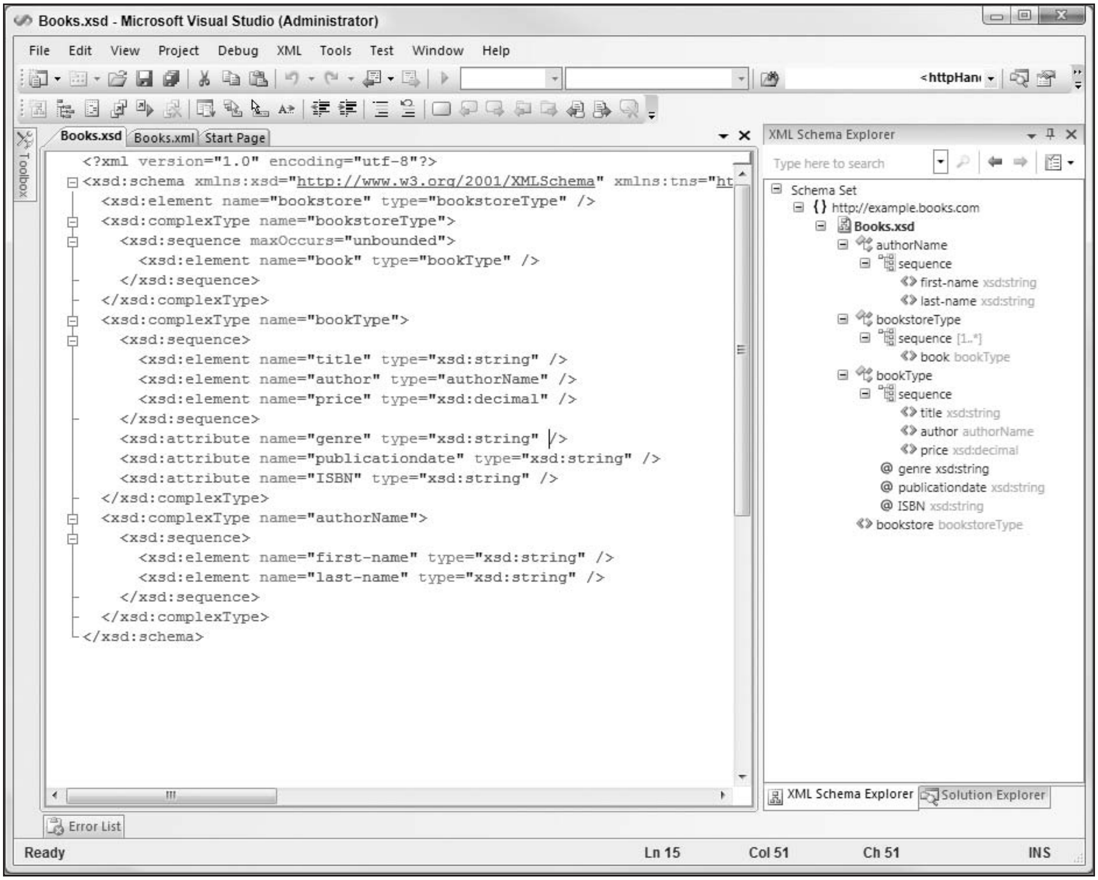

books.xsd* - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Data Tools Test Window Help
Debug
Any CPU
books.xsd*
books.xsd*
Toolbox
Solution Explorer
Properties
book
genre
publicationdate
ISBN
title
price
book_Id
bookstore_Id
bookstore
bookstore_Id
author
first-name
last-name
book_Id
Ready

Figure 10-2

When the XML Schema file from Listing 10-3 is loaded into the Visual Studio Editor, the default view in Visual Studio 2008 for standard XSDs is now the XML Editor, rather than the Dataset Designer as in Visual Studio 2005. However, if you right-click on `Books.xsd` in the Solution Explorer and select Open With, you'll get a brief warning that the DataSet Designer might have modified your schema by removing non-dataset XML. Make sure your schema is backed up and select OK, and you'll get a redesigned Dataset Designer that presents the elements and complex types in a format that is familiar if you've edited database schemas before (see Figure 10-2). This Designer view is intended to manipulate DataSets expressed as schema, but it can be a useful visualizer for many XSDs. However, as soon as you use this visualizer to edit your XSD, your document will be turned into a Microsoft DataSet. Therefore the Dataset Designer in Visual Studio 2008 is no longer suitable as a general purpose visual XML Schema Editor.

A greatly enhanced XSD Designer will be released as an add-on soon after the release of Visual Studio but wasn't yet released at the time of this writing. This new XML Schema Editor will include a Schema Explorer toolbox window that will present a comprehensive tree-view of complex schemas in a much more scalable and appropriate way than can a more traditional ER-Diagram. For more details, see Figure 10-3 or go to `http://blogs.msdn.com/xmlteam/archive/2007/08/27/announcing-ctp1-of-the-xml-schema-designer.aspx`.


Books.xsd - Microsoft Visual Studio (Administrator)
File Edit View Project Debug XML Tools Test Window Help
<httpHan
Toolbox
Books.xsd
Books.xml
Start Page
XML Schema Explorer
Type here to search
Schema Set
{} http://example.books.com
Books.xsd
authorName
sequence
first-name xsd:string
last-name xsd:string
bookstoreType
sequence [1..*]
book bookType
bookType
sequence
title xsd:string
author authorName
price xsd:decimal
@ genre xsd:string
@ publicationdate xsd:string
@ ISBN xsd:string
bookstore bookstoreType
XML Schema Explorer
Solution Explorer
Error List
Ready
Ln 15
Col 51
Ch 51
INS

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:tns="ht
  <xsd:element name="bookstore" type="bookstoreType" />
  <xsd:complexType name="bookstoreType">
    <xsd:sequence maxOccurs="unbounded">
      <xsd:element name="book" type="bookType" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="bookType">
    <xsd:sequence>
      <xsd:element name="title" type="xsd:string" />
      <xsd:element name="author" type="authorName" />
      <xsd:element name="price" type="xsd:decimal" />
    </xsd:sequence>
    <xsd:attribute name="genre" type="xsd:string" />
    <xsd:attribute name="publicationdate" type="xsd:string" />
    <xsd:attribute name="ISBN" type="xsd:string" />
  </xsd:complexType>
  <xsd:complexType name="authorName">
    <xsd:sequence>
      <xsd:element name="first-name" type="xsd:string" />
      <xsd:element name="last-name" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Figure 10-3

After you have created an XML Schema that correctly describes an XML document, you're ready to start programmatically manipulating XML. The `System.Xml` and `System.Xml.Linq` namespaces provide a number of ways to create, access, and query XML. XML Schemas provide valuable typing information for all XML consumers that are type aware.

XmlReader and XmlWriter

`XmlReader` offers a *pull-style* API over an XML document that is unique to the .NET Framework. It provides fast, forward-only, read-only access to XML documents. These documents may contain elements in multiple namespaces. `XmlReader` is actually an abstract class that other classes derive from to provide specific concrete instances like `XmlTextReader` and `XmlNodeReader`.

Things changed slightly with `XmlReader` between the .NET Framework 1.1 and 2.0 although nothing significant changed in the `XmlReader` and `XmlWriter` classes in .NET 3.5 as most of the new functionality was around LINQ. Since .NET 1.1, several convenient new methods have been added, and the way you create `XmlReader` has changed for the better. `XmlReader` has become a factory. The primary way for you to create an instance of an `XmlReader` is by using the Static/Shared `Create` method. Rather than creating concrete implementations of the `XmlReader` class, you create an instance of the `XmlReaderSettings` class and pass it to the `Create` method. You specify the features you want for your `XmlReader` object with the `XmlReaderSettings` class. For example, you might want a specialized `XmlReader` that checks the validity of an XML document with the `IgnoreWhiteSpace` and `IgnoreComments` properties pre-set. The `Create` method of the `XmlReader` class provides you with an instance of an `XmlReader` without requiring you to decide which implementation to use. You can also add features to existing `XmlReader`s by chaining instances of the `XmlReader` class with each other because the `Create` method of `XmlReader` takes another `XmlReader` as a parameter.

If you are accustomed to using the `XmlDocument` or DOM to write an entire XML fragment or document into memory, you will find using `XmlReader` to be a very different process. A good analogy is that `XmlReader` is to `XmlDocument` what the ADO ForwardOnly recordset is to the ADO Static recordset. Remember that the ADO Static recordset loads the entire results set into memory and holds it there. Certainly, you wouldn't use a Static recordset if you want to retrieve only a few values. The same basic rules apply to the `XmlReader` class. If you're going to run through the document only once, you don't want to hold it in memory; you want the access to be as fast as possible. `XmlReader` is the right decision in this case.

Listing 10-5 creates an `XmlReader` class instance and iterates forward through it, counting the number of books in the `Books.xml` document from Listing 10-1. The `XmlReaderSettings` object specifies the features that are required, rather than the actual kind of `XmlReader` to create. In this example, `IgnoreWhitespace` and `IgnoreComments` are set to `True`. The `XmlReaderSettings` object is created with these property settings and then passed to the `Create` method of `XmlReader`.

Listing 10-5: Processing XML with an XmlReader

VB

```
Imports System.IO
Imports System.Xml

Partial Class _Default
    Inherits System.Web.UI.Page
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
           Handles Me.Load
        Dim bookcount As Integer = 0
        Dim settings As New XmlReaderSettings()

        settings.IgnoreWhitespace = True
        settings.IgnoreComments = True

        Dim booksFile As String = Server.MapPath("books.xml")
        Using reader As XmlReader = XmlReader.Create(booksFile, settings)
            While (reader.Read())
                If (reader.NodeType = XmlNodeType.Element _
                        And "book" = reader.LocalName) Then
                    bookcount += 1
                End If
            End While
        End Using
        Response.Write(String.Format("Found {0} books!", bookcount))
    End Sub
End Class
```

C#

```
using System;
using System.IO;
using System.Xml;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        int bookcount = 0;
        XmlReaderSettings settings = new XmlReaderSettings();

        settings.IgnoreWhitespace = true;
        settings.IgnoreComments = true;

        string booksFile = Server.MapPath("books.xml");
        using (XmlReader reader = XmlReader.Create(booksFile, settings))
        {
            while (reader.Read())
            {
                if (reader.NodeType == XmlNodeType.Element &&
                        "book" == reader.LocalName)
                {
                    bookcount++;
                }
            }
        }
        Response.Write(String.Format("Found {0} books!", bookcount));
    }
}
```

Notice the use of the `XmlReader.Create` method in Listing 10-5. You may be used to creating concrete implementations of an `XmlReader`, but if you try this technique, you should find it much more flexible because you can reuse the `XmlReaderSettings` objects in the creation of other instances of `XmlReader`. `XmlReader` implements `IDisposable`, so the `Using` keyword is correct in both VB and C#.

In Listing 9-5 the `Books.xml` file is in the same directory as this ASPX page, so a call to `Server.MapPath` gets the complete path to the XML file. The filename with full path is then passed into `XmlReader.Create`, along with the `XmlReaderSettings` instance from a few lines earlier.

The `Read` method continues to return `true` if the node was read successfully. It will return `false` when no more nodes are left to read. From the point of view of an `XmlReader`, everything is a node including white space, comments, attributes, elements, and end elements. If Listing 10-5 had simply spun through the `while` loop incrementing the `bookcount` variable each time `reader.LocalName` equaled `book`, the final value for `bookcount` would have been six. You would have counted both the beginning `book` tag and the ending `book` tag. Consequently, you have to be more explicit, and ensure that the `if` statement is modified to check not only the `LocalName` but also the `NodeType`.

The `Reader.LocalName` property contains the non–namespace qualified name of that node. The `Reader.Name` property is different and contains the fully qualified name of that node including namespace. The `Reader.LocalName` property is used in the example in Listing 10-5 for simplicity and ease. You'll hear more about namespaces a little later in the chapter.

Using XDocument Rather Than XmlReader

The `System.Xml.Linq` namespace introduces a new `XDocument` class that presents a much friendlier face than `XmlDocument` while still allowing for interoperability with `XmlReaders` and `XmlWriters`. Listing10-5q accomplishes the same thing as Listing 10-5 but uses `XDocument` instead. The `XDocument` is loaded just like an `XmlDocument` but the syntax for retrieving the elements we want is significantly different.

The syntax for this query is very clean, but slightly reversed from what you may be used to if you've used T-SQL. Rather than `select`...`from`, we're using the standard LINQ `from`...`select` syntax. We ask the `booksXML` `XDocument` for all its `book` descendants, and they are selected into the `book` range variable. The value of all the book `title` elements is then selected into the `books` variable.

VB takes the opportunity in Visual Studio 2008 to distinguish itself considerably from C# by including a number of bits of ''syntactic sugar,'' which makes the experience of working with Visual Basic and XML more integrated. Notice the use of the `Imports` keyword to declare an XML namespace, as well as the use of ''...<>'' to indicate the method call to `Descendants` and ''.<>'' to call `Elements`. This extraordinary level of XML integration with the compiler really makes working with XML in VB a joy — and this is a C# lover speaking.

Listing 10-5q: Processing XML with a XDocument

VB

```
Imports System.IO
Imports System.Xml
Imports System.Linq
Imports System.Xml.Linq
Imports <xmlns:b="http://example.books.com">

Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
           Handles Me.Load

        Dim booksXML = XDocument.Load(Server.MapPath("books.xml"))
        Dim books = From book In booksXML...<b:book> Select book.<b:title>.Value

        Response.Write(String.Format("Found {0} books!", books.Count()))
    End Sub
End Class
```

C#

```
using System;
using System.IO;
using System.Linq;
using System.Xml.Linq;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        XDocument booksXML = XDocument.Load(Server.MapPath("books.xml"));
        var books = from book in
                    booksXML.Descendants("{http://example.books.com}book")
                    select book.Element("{http://example.books.com}title").Value;
        Response.Write(String.Format("Found {0} books!", books.Count()));
    }
}
```

In both the C# and VB examples, we take advantage of the implicit typing by not indicating the return type in the call to `XDocument.Decendants`. In VB we use `Dim books` and in C# we use `var books`. In this example, because we are using the `from`...`select` syntax to select our books from the `booksXml` object, the type of the variable `books` is `System.Linq.Enumerable.SelectIterator`, which is ultimately `IEnumerable`. The `count` method is added by LINQ as an extension method, allowing us to retrieve the number of books.

Notice also that the `Books.xml` document has a namespace of `http://examples.books.com`, so elements with this namespace are included in the query using the LINQ for XML format of `namespaceelement`. In later examples we'll use the `XNamespace` object to make the C# syntax slightly cleaner.

Using Schema with XmlTextReader

The code in Listing 10-5 reads any XML document regardless of its schema, and if the document contains an element named `book`, the code counts it. If this code is meant to count books of a particular schema type only, specifically the books from the `Books.xml` file, it should be validated against the `Books.xsd` schema.

Now modify the creation of the `XmlReader` class from Listing 10-5 to validate the `XmlDocument` against the XML Schema used earlier in the chapter. Note that the `XmlValidatingReader` class is now considered obsolete because all reader creation is done using the `Create` method of the `XmlReader` class.

Listing 10-6 shows a concrete example of how easy it is to add schema validation to code using `XmlReaderSettings` and the `XmlReader` `Create` method.

Listing 10-6: Validating XML with an XmlReader against an XML Schema

VB

```
Imports System.Xml.Schema

Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles Me.Load
    Dim bookcount As Integer = 0
    Dim settings As New XmlReaderSettings()
    Dim booksSchemaFile As String = Server.MapPath("books.xsd")

    settings.Schemas.Add(Nothing, XmlReader.Create(booksSchemaFile))
    settings.ValidationType = ValidationType.Schema
    settings.ValidationFlags = _
        XmlSchemaValidationFlags.ReportValidationWarnings

    AddHandler settings.ValidationEventHandler, _
    AddressOf settings_ValidationEventHandler

    settings.IgnoreWhitespace = True
    settings.IgnoreComments = True

    Dim booksFile As String = Server.MapPath("books.xml")
    Using reader As XmlReader = XmlReader.Create(booksFile, settings)
        While (reader.Read())
            If (reader.NodeType = XmlNodeType.Element _
                    And "book" = reader.LocalName) Then
                bookcount += 1
            End If
        End While
    End Using
    Response.Write(String.Format("Found {0} books!", bookcount))
End Sub

Sub settings_ValidationEventHandler(ByVal sender As Object, _
        ByVal e As System.Xml.Schema.ValidationEventArgs)
    Response.Write(e.Message)
End Sub
```

C#

```
using System.Xml.Schema;

protected void Page_Load(object sender, EventArgs e)
{
    int bookcount = 0;
    XmlReaderSettings settings = new XmlReaderSettings();

    string booksSchemaFile = Server.MapPath("books.xsd");

    settings.Schemas.Add(null, XmlReader.Create(booksSchemaFile));
```

```
        settings.ValidationType = ValidationType.Schema;
        settings.ValidationFlags =

        XmlSchemaValidationFlags.ReportValidationWarnings;
        settings.ValidationEventHandler +=
                new ValidationEventHandler(settings_ValidationEventHandler);

        settings.IgnoreWhitespace = true;
        settings.IgnoreComments = true;

        string booksFile = Server.MapPath( "books.xml");
        using (XmlReader reader = XmlReader.Create(booksFile, settings))
        {
            while (reader.Read())
            {
                if (reader.NodeType == XmlNodeType.Element &&
                        "book" == reader.LocalName)
                {
                    bookcount++;
                }
            }
        }
        Response.Write(String.Format("Found {0} books!", bookcount));
    }

    void settings_ValidationEventHandler(object sender,
            System.Xml.Schema.ValidationEventArgs e)
    {
        Response.Write(e.Message);
    }
```

When validating XML, the validator uses the `schemaLocation` hint found in the XML instance document. If an XML instance document does not contain enough information to find an XML Schema, the instance document expects an `XmlSchemaSet` object on the `XmlReaderSettings` object. In the interest of being explicit, Listing 10-6 shows this technique. The `XmlReaderSettings` object has a `Schemas` collection available as a property and many overloads for the `Add` method. This listing passes `null` into the `Add` method as the first parameter, indicating that the `targetNamespace` is specified in the schema. Optionally, XML documents can also contain their schemas inline.

The validator needs a way to let you know when validation problems occur. The `XmlReaderSettings` object has a validation event handler that notifies you as validation events occur. Listing 10-6 also includes a handler for the validation event that writes the message to the browser.

Validating Against a Schema Using an XDocument

Much of `System.Xml.Linq` is ''bridged'' to `System.Xml` by using extension methods. For example, the `XDocument` class has an extension `Validate` method that takes a standard `System.Xml.Schema.XmlSchemaSet` as a parameter, allowing us to validate an `XDocument` against an XML Schema.

In Listing 10-6q, the `XmlSchemaSet` is loaded in the standard way, and then passed into the `XDocument`'s validate method.

Listing 10-6q: Validating XML with a LINQ XDocument against an XML Schema

VB

```
Imports System
Imports System.Xml
Imports System.Linq
Imports System.Xml.Linq
Imports System.Xml.Schema

Imports <xmlns:b="http://example.books.com">

Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
           Handles Me.Load
        Dim schemas = New XmlSchemaSet()
        schemas.Add(Nothing, XmlReader.Create(Server.MapPath("books.xsd")))
        Dim booksXML = XDocument.Load(Server.MapPath("books.xml"))
        booksXML.Validate(schemas, AddressOf ValidationEventHandler, True)

        Dim books = From book In booksXML...<b:book> _
                    Select book.<b:title>.Value

        Response.Write(String.Format("Found {0} books!", books.Count()))
    End Sub

    Sub ValidationEventHandler(ByVal sender As Object, _
            ByVal e As System.Xml.Schema.ValidationEventArgs)
        Response.Write(e.Message)
    End Sub
End Class
```

C#

```
using System;
using System.Xml;
using System.Xml.Linq;
using System.Linq;
using System.Xml.Schema;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string booksSchemaFile = Server.MapPath("books.xsd");
        string booksFile = Server.MapPath("books.xml");

        XmlSchemaSet schemas = new XmlSchemaSet();
        schemas.Add(null, XmlReader.Create(booksSchemaFile));
        XDocument booksXML = XDocument.Load(booksFile);
        booksXML.Validate(schemas, (senderParam, eParam) =>
                                   {
                                     Response.Write(eParam.Message);
                                   }, true);
```

```
            XNamespace ns = "http://example.books.com";
            var books = from book in booksXML.Descendants(ns + "book")
                        select book.Element(ns + "title").Value;
            Response.Write(String.Format("Found {0} books!", books.Count()));
        }
    }
```

Notice the unique syntax for an anonymous event handler in the C# example in Listing 10-6q. Rather than creating a separate method and passing it into the call to `Validate`, C# 3.0 programmers can pass the method body anonymously in as a parameter to the `Validate` method. The `(param1, param2) => { method }` syntax can be a bit jarring initially, but it makes for much tidier code.

Including NameTable Optimization

`XmlReader` internally uses a `NameTable` that lists all the known elements and attributes with namespaces that are used in that document. This process is called *atomization*-literally meaning that the XML document is broken up into its atomic parts. There's no need to store the string `book` more than once in the internal structure if you can make `book` an object reference that is held in a table with the names of other elements.

Although this is an internal implementation detail, it is a supported and valid way that you can measurably speed up your use of XML classes, such as `XmlReader` and `XmlDocument`. You add name elements to the `NameTable` that you know will be in the document. Listings 9-5 and 9-6 use string comparisons to compare a string literal with `reader.LocalName`. These comparisons can also be optimized by turning them into object reference comparisons that are many, many times faster. Additionally, an XML `NameTable` can be shared across multiple instances of `System.Xml` classes and even between `XmlReaders` and `XmlDocuments`. This topic is covered shortly.

Because you are counting `book` elements, create a `NameTable` including this element (`book`), and instead of comparing string against string, compare object reference against object reference, as shown in Listing 10-7.

Listing 10-7: Optimizing XmlReader with a NameTable

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles Me.Load
    Dim bookcount As Integer = 0
    Dim settings As New XmlReaderSettings()
    Dim nt As New NameTable()
    Dim book As Object = nt.Add("book")

    settings.NameTable = nt
    Dim booksSchemaFile As String = _
        Path.Combine(Request.PhysicalApplicationPath, "books.xsd")
    settings.Schemas.Add(Nothing, XmlReader.Create(booksSchemaFile))
    settings.ValidationType = ValidationType.Schema
    settings.ValidationFlags = _
    XmlSchemaValidationFlags.ReportValidationWarnings

    AddHandler settings.ValidationEventHandler, _
        AddressOf settings_ValidationEventHandler
    settings.IgnoreWhitespace = True
    settings.IgnoreComments = True

    Dim booksFile As String = _
    Path.Combine(Request.PhysicalApplicationPath, "books.xml")
    Using reader As XmlReader = XmlReader.Create(booksFile, settings)
        While (reader.Read())
            If (reader.NodeType = XmlNodeType.Element _
                And book.Equals(reader.LocalName)) Then
                'A subtle, but significant change!
                bookcount += 1
            End If
        End While
    End Using
    Response.Write(String.Format("Found {0} books!", bookcount))
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    int bookcount = 0;
    XmlReaderSettings settings = new XmlReaderSettings();
    NameTable nt = new NameTable();
    object book = nt.Add("book");

    settings.NameTable = nt;
    string booksSchemaFile = Path.Combine(Request.PhysicalApplicationPath,
        "books.xsd");

    settings.Schemas.Add(null, XmlReader.Create(booksSchemaFile));
    settings.ValidationType = ValidationType.Schema;
    settings.ValidationFlags =
        XmlSchemaValidationFlags.ReportValidationWarnings;

    settings.ValidationEventHandler +=
        new ValidationEventHandler(settings_ValidationEventHandler);

    settings.IgnoreWhitespace = true;
    settings.IgnoreComments = true;

    string booksFile = Path.Combine(Request.PhysicalApplicationPath, "books.xml");
    using (XmlReader reader = XmlReader.Create(booksFile, settings))
    {
        while (reader.Read())
        {
            if (reader.NodeType == XmlNodeType.Element &&
                book.Equals(reader.LocalName)) //A subtle, but significant change!
            {
                bookcount++;
            }
        }
    }
    Response.Write(String.Format("Found {0} books!", bookcount));
}
```

The `NameTable` is added to the `XmlSettings` object and the `Add` method of the `NameTable` returns an object reference to the just-added atom that is stored, in this case, in an object reference named `book`. The `book` reference is then used later to make a comparison to the `reader.LocalName` property. We specifically chose to use the `Equals` method that is present on all objects within that .NET Framework in order to emphasize that this is specifically an object identity check for equality. These two objects are either the same identical atoms or they are not. The `book` object that is returned from the `Add` method on the `NameTable` is the identical object that the reader uses when parsing the `book` element from the `Books.xml` XML document.

In the example of Listing 10-7, in which you count a very small number of books, you probably won't have a measurable performance gain. However, for larger XML documents that approach sizes of 1 MB, you may see performance gains of as much as 10 to 15 percent — especially for the involved calculations and manipulations of `XmlReader`. Additionally, because the `NameTable` is cached within the `XmlReaderSettings` object, that `NameTable` is reused when the `XmlReaderSettings` object is reused for other `System.Xml` objects. This creates additional potential performance gains.

Retrieving .NET CLR Types from XML

It is considerably simpler to retrieve CLR types from an `XmlReader` than it was previously in the 1.*x* Framework. If you've used SQL Server data reader objects before, retrieving data types from `XmlReader` should feel very familiar. Previously the Framework used a helper class called `XmlConvert`. When combined with the `ReadElementString` method on `XmlReader`, this helper class retrieved a strong, simple type, as shown in the following code:

```
//Retrieving a double from an XmlReader in the .NET Framework 1.1
Double price = XmlConvert.ToDouble(reader.ReadElementString());
//Has been replaced by and improved in the .NET Framework 2.0
Double price = reader.ReadElementContentAsDouble();
```

You can see the removal of the unnecessary double method call results in much cleaner and easier-to-read code. Listing 10-8 adds not only the counting of books but also prints the total price of all books using `ReadElementContentAs` when your `XmlReader` is currently on an element, or `ReadContentAs` if on text content. If schema information is available to the reader, `ReadElementContentAsObject` returns the value directly as, in this case, a decimal. If the reader does not have any schema information, it attempts to convert the string to a decimal. A whole series of `ReadElementContentAs` and `ReadContentAs` methods, including `ReadElementContentAsBoolean` and `ReadElementContentAsInt`, are available. Note that the code specific to `XmlSchema` has been removed from Listing 10-8 in the interest of brevity.

Listing 10-8: Using XmlReader.ReadElementContentAs

VB

```
Dim bookcount As Integer = 0
Dim booktotal As Decimal = 0
Dim settings As New XmlReaderSettings()
Dim nt As New NameTable()
Dim book As Object = nt.Add("book")
Dim price As Object = nt.Add("price")

settings.NameTable = nt

Dim booksFile As String = _
```

```
Path.Combine(Request.PhysicalApplicationPath, "books.xml")
Using reader As XmlReader = XmlReader.Create(booksFile, settings)
    While (reader.Read())
        If (reader.NodeType = XmlNodeType.Element _
            And book.Equals(reader.LocalName)) Then
            bookcount += 1
        End If
        If (reader.NodeType = XmlNodeType.Element _
            And price.Equals(reader.LocalName)) Then
            booktotal += reader.ReadElementContentAsDecimal()
        End If
    End While
End Using

Response.Write(String.Format("Found {0} books that total {1:C}!", _
    bookcount, booktotal))
```

C#

```
int bookcount = 0;
decimal booktotal = 0;
XmlReaderSettings settings = new XmlReaderSettings();
string booksSchemaFile = Path.Combine(Request.PhysicalApplicationPath, "books.xsd");
NameTable nt = new NameTable();
object book = nt.Add("book");
object price = nt.Add("price");

settings.NameTable = nt;

string booksFile = Path.Combine(Request.PhysicalApplicationPath, "books.xml");

using (XmlReader reader = XmlReader.Create(booksFile, settings))
{
    while (reader.Read())
    {
        if (reader.NodeType == XmlNodeType.Element &&
            book.Equals(reader.LocalName))//A subtle, but significant change!
        {
            bookcount++;
        }
        if (reader.NodeType == XmlNodeType.Element &&
        price.Equals(reader.LocalName))
        {
            booktotal +=
                reader.ReadElementContentAsDecimal ();
        }
    }
}

Response.Write(String.Format("Found {0} books that total {1:C}!",
    bookcount, booktotal));
```

The `booktotal` variable from Listing 10-8 is strongly typed as a decimal so that, in the `String.Format` call, it can be formatted as currency using the formatting string `{1:C}`. This results in output from the browser similar to the following:

```
Found 3 books that total $30.97!
```

ReadSubtree and XmlSerialization

Not only does `XmlReader` help you retrieve simple types from XML, it can help you retrieve more complicated types using XML serialization and `ReadSubtree`.

XML serialization allows you to add attributes to an existing class that give hints to the XML serialization on how to represent an object as XML. XML serialization serializes only the public properties of an object, not the private ones.

When you create an `XmlSerializer`, a `Type` object is passed into the constructor, and the `XmlSerializer` uses reflection to examine whether the object can create a temporary assembly that knows how to read and write this particular object as XML. The `XmlSerializer` uses a concrete implementation of `XmlReader` internally to serialize these objects.

Instead of retrieving the author's first name and last name using `XmlReader.ReadAsString`, Listing 10-10 below uses `ReadSubtree` and a new strongly typed `Author` class that has been marked up with XML serialization attributes, as shown in Listing 10-9. `ReadSubtree` ''breaks off'' a new `XmlReader` at the current location, and that `XmlReader` is passed to an `XmlSerializer` and a complex type is created. The `Author` class includes `XmlElement` attributes that indicate, for example, that although there is a property called `FirstName`, it should be serialized and deserialized as ''first-name.''

Listing 10-9: An Author class with XML serialization attributes matching Books.xsd

VB

```
Imports System.Xml.Serialization
<XmlRoot(ElementName:="author", _
Namespace:="http://example.books.com")> Public Class Author
    <XmlElement(ElementName:="first-name")> Public FirstName As String
    <XmlElement(ElementName:="last-name")> Public LastName As String
End Class
```

C#

```
using System.Xml.Serialization;
[XmlRoot(ElementName = "author", Namespace = "http://example.books.com")]
public class Author
{
    [XmlElement(ElementName = "first-name")]
    public string FirstName;

    [XmlElement(ElementName = "last-name")]
    public string LastName;
}
```

Next, this `Author` class is used along with `XmlReader.ReadSubtree` and `XmlSerializer` to output the names of each book's author. Listing 10-10 shows just the additional statements added to the `While` loop.

Listing 10-10: Reading author instances from an XmlReader using XmlSerialization

VB

```
'Create factory early
Dim factory As New XmlSerializerFactory
```

```
Using reader As XmlReader = XmlReader.Create(booksFile, settings)
    While (reader.Read())
        If (reader.NodeType = XmlNodeType.Element _
            And author.Equals(reader.LocalName)) Then

            'Then use the factory to create and cache serializers
            Dim xs As XmlSerializer = factory.CreateSerializer(GetType(Author))
            Dim a As Author = CType(xs.Deserialize(reader.ReadSubtree), Author)
            Response.Write(String.Format("Author: {1}, {0}<BR/>", _
                a.FirstName, a.LastName))
         End If
     End While
End Using
```

C#

```
//Create factory early
XmlSerializerFactory factory = new XmlSerializerFactory();

using (XmlReader reader = XmlReader.Create(booksFile, settings))
{
    while (reader.Read())
    {
        if (reader.NodeType == XmlNodeType.Element &&
            author.Equals(reader.LocalName))
        {
            //Then use the factory to create and cache serializers
            XmlSerializer xs = factory.CreateSerializer(typeof(Author));
            Author a = (Author)xs.Deserialize(reader.ReadSubtree());
            Response.Write(String.Format("Author: {1}, {0}<BR/>",
                a.FirstName, a.LastName));
        }
    }
}
```

The only other addition to the code, as you can guess, is the `author` object atom (used only in the `Equals` statement) that is added to the `NameTable` just as the book and price were, via `Dim author As Object = nt.Add("author")`.

When you create an `XmlSerializer` instance for a specific type, the Framework uses reflection to create a temporary type-specific assembly to handle serialization and deserialization. The .NET Framework 2.0 includes a new `XmlSerializerFactory` that automatically handles caching of these temporary assemblies. This small factory provides an important layer of abstraction that allows you to structure your code in a way that is convenient without worrying about creating `XmlSerializer` instances ahead of time.

Creating CLR Objects from XML with LINQ to XML

While there isn't a direct bridge between the `XmlSerializer` and `System.Xml.Linq`, there is a very clean way of creating CLR objects within the LINQ to XML syntax. This syntax can also be a little more flexible and more forgiving than the traditional `XmlSerializer`, as shown in Listing 10-10q.

At the time of this writing there is talk of a future bridge technology that will bring strongly typed objects and XML together. It's currently called LINQ to XSD and you can find more information on the XMLTeam's blog at `http://blogs.msdn.com/xmlteam/`.

Listing 10-10q: Reading author instances via LINQ to XML

VB

```
Dim booksXML = XDocument.Load(Server.MapPath("books.xml"))

Dim authors = From book In booksXML...<books:book> Select New Author _
    With {.FirstName = book.<books:author>.<books:first-name>.Value, _
          .LastName = book.<books:author>.<books:last-name>.Value}

    For Each a As Author In authors
    Response.Write(String.Format("Author: {1}, {0}<BR/>", a.FirstName, a.LastName))
Next
```

C#

```
XDocument booksXML = XDocument.Load(Server.MapPath("books.xml"));
XNamespace ns = "http://example.books.com";

var authors = from book in booksXML.Descendants(ns + "author")
            select new Author
            {
                FirstName = book.Element(ns + "first-name").Value,
                LastName = book.Element(ns + "last-name").Value
            };
foreach (Author a in authors)
{
    Response.Write(String.Format("Author: {1}, {0}<BR/>", a.FirstName, a.LastName));
}
```

Again, note the unique syntax in the VB example, where ''...'' is used rather than ''descendants.'' On the C# side, notice how cleanly a new Author object is created with the `select new` syntax, and within the curly braces that new `Author` object has its property values copied over from the XML elements. The VB example assumes the same namespace `Imports` statement as seen in the previous example in Listing 10-6q.

Creating XML with XmlWriter

`XmlWriter` works exactly like `XmlReader` except in reverse. It's very tempting to use string concatenation to quickly create XML documents or fragments of XML, but you should resist the urge! Remember that the whole point of XML is the representation of the InfoSet, not the angle brackets. If you concatenate string literals together with `StringBuilder` to create XML, you are dropping below the level of the InfoSet to the implementation details of the format. Tell yourself that XML documents are not strings!

Most people find it helpful (as a visualization tool) to indent the method calls to the `XmlWriter` with the same structure as the resulting XML document. However, VB in Visual Studio is much more aggressive than C# in keeping the code indented a specific way. It does not allow this kind of artificial indentation unless Smart Indenting is changed to either Block or None by using Tools ➪ Options ➪ Text Editor ➪ Basic ➪ Tabs.

`XmlWriter` also has a settings class called, obviously, `XmlWriterSettings`. This class has options for indentation, new lines, encoding, and XML conformance level. Listing 10-11 uses `XmlWriter` to create a `bookstore` XML document and output it directly to the ASP.NET `Response.OutputStream`. All the HTML tags in the ASPX page must be removed in order for the XML document to be output correctly. Another way to output XML easily is with an ASHX `HttpHandler`.

The unusual indenting in Listing 10-11 is significant and very common when using `XmlWriter`. It helps the programmer visualize the hierarchical structure of an XML document.

Listing 10-11: Writing out a bookstore with XmlWriter

Default.aspx — C#

```
<%@ Page Language="C#" codefile="Default.aspx.cs" Inherits="Default_aspx" %>
```

Default.aspx — VB

```
<%@ Page Language="VB" codefile="Default.aspx.vb" Inherits="Default_aspx" %>
```

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles Me.Load

Dim price As Double = 49.99
Dim publicationdate As New DateTime(2005, 1, 1)
Dim isbn As String = "1-057-610-0"
Dim a As New Author()
a.FirstName = "Scott"
a.LastName = "Hanselman"

Dim settings As New XmlWriterSettings()
settings.Indent = True
settings.NewLineOnAttributes = True

Response.ContentType = "text/xml"

Dim factory As New XmlSerializerFactory()

Using writer As XmlWriter = XmlWriter.Create(Response.OutputStream, settings)

    'Note the artificial, but useful, indenting
    writer.WriteStartDocument()
        writer.WriteStartElement("bookstore")
            writer.WriteStartElement("book")
                writer.WriteStartAttribute("publicationdate")
                    writer.WriteValue(publicationdate)
                writer.WriteEndAttribute()
                writer.WriteStartAttribute("ISBN")
                    writer.WriteValue(isbn)
                writer.WriteEndAttribute()
                writer.WriteElementString("title", "ASP.NET 2.0")
                writer.WriteStartElement("price")
                    writer.WriteValue(price)
                writer.WriteEndElement() 'price
                Dim xs As XmlSerializer = _
                    factory.CreateSerializer(GetType(Author))
                xs.Serialize(writer, a)
            writer.WriteEndElement() 'book
        writer.WriteEndElement() 'bookstore
    writer.WriteEndDocument()
End Using

End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
Double price = 49.99;
DateTime publicationdate = new DateTime(2005, 1, 1);
String isbn = "1-057-610-0";
Author a = new Author();
a.FirstName = "Scott";
a.LastName = "Hanselman";

XmlWriterSettings settings = new XmlWriterSettings();
settings.Indent = true;
settings.NewLineOnAttributes = true;

Response.ContentType = "text/xml";

XmlSerializerFactory factory = new XmlSerializerFactory();

using (XmlWriter writer =
        XmlWriter.Create(Response.OutputStream, settings))
{
    //Note the artificial, but useful, indenting
    writer.WriteStartDocument();
        writer.WriteStartElement("bookstore");
            writer.WriteStartElement("book");
                writer.WriteStartAttribute("publicationdate");
                    writer.WriteValue(publicationdate);
                writer.WriteEndAttribute();
                writer.WriteStartAttribute("ISBN");
                    writer.WriteValue(isbn);
                writer.WriteEndAttribute();
                writer.WriteElementString("title", "ASP.NET 2.0");
                writer.WriteStartElement("price");
                    writer.WriteValue(price);
                writer.WriteEndElement(); //price
                XmlSerializer xs = factory.CreateSerializer(typeof(Author));
                xs.Serialize(writer, a);
            writer.WriteEndElement(); //book
        writer.WriteEndElement(); //bookstore
    writer.WriteEndDocument();
}

}
```

The `Response.ContentType` in Listing 10-11 is set to `"text/xml"` to indicate to Internet Explorer that the result is XML. An `XmlSerializer` is created in the middle of the process and serialized directly to `XmlWriter`. The `XmlWriterSettings.Indent` property includes indentation that makes the resulting XML document more palatable for human consumption. Setting both this property and `NewLineOnAttributes` to `false` results in a smaller, more compact document.

Creating XML with LINQ for XML

Listing 10-11q accomplishes the same thing as Listing 10-11 but with LINQ for XML. It won't be as lightening fast as `XmlWriter`, but it will still be extremely fast and it is very easy to read.

Notice the dramatic difference between the VB and C# examples. The VB example uses a new VB9 compiler feature called Xml Literals, The XML structure is typed directly in the code without any quotes. It's not a string. The compiler will turn the XML literal into a tree of XElements similar to the syntax used in the C# example. The underlying LINQ to XML technology is the same — only the syntax differs. One syntax thing to note in the VB example is the exclusion of double-quotes around the attributes for `publicationdate` and `ISBN`. The quotes are considered part of the serialization format of the attribute value and they will be added automatically for you when the final XML is created.

Listing 10-11q: Writing out a bookstore with XElement trees

VB

```
Imports System.Xml
Imports System.Xml.Linq

Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
      Handles Me.Load
        Dim price As Double = 49.99
        Dim publicationdate As New DateTime(2005, 1, 1)
        Dim isbn As String = "1-057-610-0"
        Dim a As New Author()
        a.FirstName = "Scott"
        a.LastName = "Hanselman"

        Response.ContentType = "text/xml"

        Dim books = <bookstore xmlns="http://examples.books.com">
                     <book publicationdate=<%= publicationdate %> ISBN=<%= isbn %>>
                            <title>ASP.NET 2.0</title>
                            <price><%= price %></price>
                            <author>
                                <first-name><%= a.FirstName %></first-name>
                                <last-name><%= a.LastName %></last-name>
                            </author>
                     </book>
                 </bookstore>

        Response.Write(books)
    End Sub
End Class
```

C#

```
using System;
using System.Xml.Linq;

public partial class _Default : System.Web.UI.Page
```

```
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Double price = 49.99;
        DateTime publicationdate = new DateTime(2005, 1, 1);
        String isbn = "1-057-610-0";
        Author a = new Author();
        a.FirstName = "Scott";
        a.LastName = "Hanselman";

        Response.ContentType = "text/xml";
        XNamespace ns = "http://example.books.com";
        XDocument books = new XDocument(
            new XElement(ns + "bookstore",
                new XElement(ns + "book",
                    new XAttribute("publicationdate", publicationdate),
                    new XAttribute("ISBN", isbn),
                    new XElement(ns + "title", "ASP.NET 2.0 Book"),
                    new XElement(ns + "price", price),
                    new XElement(ns + "author",
                        new XElement(ns + "first-name", a.FirstName),
                        new XElement(ns + "last-name", a.LastName)
                        )
                    )
                )
            );
        Response.Write(books);
    }
}
```

Even though the C# example appears on multiple lines, it could be all on one line as it is one single expression. The constructors for `XElement` and `XDocument` take an array of parameters that is arbitrarily long, allowing us to create an XML Document structure more declaratively. If you compare this listing to Listing 10-11, you'll notice that the indentation in the `XmlWriter` example makes the sample more readable, but doesn't affect document structure. However, with `XDocument/XElement` declarations, the document structure is expressed by how the objects are nested as they are passed into each other's constructors.

There was a ''Paste XML as XLinq'' feature that didn't make it into the released product but was shipped as a Sample. This add-in adds a menu item to your Visual Studio 2008 Edit Menu that will take XML from the clipboard and paste it into the editor as an XElement expression. You'll find it from within Visual Studio via the Help ⇨ Samples ⇨ Visual C# Samples folder. The sample is called `PasteXmlAsLinq` and can be compiled and copied to any Visual Studio AddIns folder.

Bridging XmlSerializer and LINQ to XML

There isn't a direct bridge between `XmlSerialization` and the new LINQ classes; both `XDocument` and `XElement` include a `CreateWriter` method that returns an `XmlWriter`. When that returned `XmlWriter` has its `Close` method called, all the generated XML is turned into XElements and added to the parent `XDocument` or `XElement`. This allows us to mix XElements and XmlSerialization techniques within a single expression. While this could be considered an unusual use case, XmlSerialization happens often, and it's useful to point out how well the new 3.5 classes can get along with the 2.0 classes.

This C# example is the same as Listing 10-11 except it creates an Extension Class to extend `XmlSerializer` with a new `SerializeAsXElement` method that returns an `XElement` containing the result of the serialization. First, the extension:

```
static class XmlSerializerExtension
{
    public static XElement SerializeAsXElement(this XmlSerializer xs, object o)
    {
        XDocument d = new XDocument();
        using (XmlWriter w = d.CreateWriter())
        {
            xs.Serialize(w, o);
        }
        XElement e = d.Root;
        e.Remove();
        return e;
    }
}
```

Notice that the class and method are `static`, and the `this` keyword refers to the class being extended, in this case `XmlSerializer`. An `XDocument` is created and an `XmlWriter` is returned. The `XmlSerializer` then serializes the object to this `XmlWriter`. Then there is a little nuance as the root element is removed from the document. This avoids cloning of the returned element during any subsequent uses within a functional construction.

Now our new extension method on the `XmlSerializer` can be called within the middle of the XElement functional construction, as shown here:

```
XmlSerializer xs = new XmlSerializer(typeof(Author));
XDocument books = new XDocument(
    new XElement(ns + "bookstore",
        new XElement(ns + "book",
            new XAttribute("publicationdate", publicationdate),
            new XAttribute("ISBN", isbn),
            new XElement(ns + "title", "ASP.NET 2.0 Book"),
            new XElement(ns + "price", price),
            xs.SerializeAsXElement(a)
            )
        )
    );
```

The resulting XML is identical to the output of Listing 10-11. Now you've got three different ways to create XML, first with a Writer, second with LINQ to XML, and third as a hybrid of XmlSerialization and LINQ. There are other combinations that you can come up with that maximize existing code reuse and ease of development.

Improvements for XmlReader and XmlWriter in 2.0

A few helper methods and changes make using `XmlReader` and `XmlWriter` even simpler in the .NET Framework 2.0:

- ❑ `ReadSubtree`: This method reads the current node of an `XmlReader` and returns a new `XmlReader` that traverses the current node and all its descendants. It allows you to chop off a portion of the XML InfoSet and process it separately.

- `ReadToDescendant` and `ReadToNextSibling`: These two methods provide convenient ways to advance the `XmlReader` to specific elements that appear later in the document.
- `Dispose`: `XmlReader` and `XmlWriter` both implement `IDisposable`, which means that they support the `Using` keyword. `Using`, in turn, calls `Dispose`, which calls the `Close` method. These methods are now less problematic because you no longer have to remember to call `Close` to release any resources. This simple but powerful technique has been used in the listings in this chapter.

XmlDocument and XPathDocument

In the .NET Framework 1.1, the `XmlDocument` was one of the most common ways to manipulate XML. It is similar to using a static ADO recordset because it parses and loads the entire `XmlDocument` into memory. Often the `XmlDocument` is the first class a programmer learns to use and, consequently, as a solution it becomes the hammer in his toolkit. Unfortunately, not every kind of XML problem is a nail. `XmlDocuments` have been known to use many times their file size in memory. Often an `XmlDocument` is referred to as the DOM or Document Object Model. The `XmlDocument` is compliant with the W3C DOM implementation and should be familiar to anyone who has used a DOM implementation.

Problems with the DOM

There are a number of potential problems with the `XmlDocument` class in the .NET Framework. The data model of the `XmlDocument` is very different from other XML query languages such as XSLT and XPath. The `XmlDocument` is editable and provides a familiar API for those who used MSXML in Visual Basic 6. Often, however, people use the `XmlDocument` to search for data within a larger document, but the `XmlDocument` isn't designed for searching large amounts of information. The `XPathDocument` is read-only and optimized for XPath queries or XPath-heavy technologies such as XSLT. In .NET Framework 2.0, the `XPathDocument` is much, much faster than the `XmlDocument` for loading and querying XML.

The `XPathDocument` is very focused around the InfoSet because it has a much-optimized internal structure. Be aware, however, that it does throw away insignificant white spaces and CDATA sections, so it is not appropriate if you want the `XPathDocument` to maintain the identical number of bytes that you originally created. However, if you're focused more on the set of information that is contained within your document, you can be assured that the `XPathDocument` contains everything that your source document contains.

A rule of thumb for querying data is that you should use the `XPathDocument` instead of the `XmlDocument` — except in situations where you must maintain compatibility with previous versions of the .NET Framework. The new `XPathDocument` supports all the type information from any associated XML Schema and supports the schema validation via the `Validate` method. The `XPathDocument` lets you load XML documents to URLs, files, or streams. The `XPathDocument` is also the preferred class to use for the XSLT transformations covered later in this chapter.

XPath, the XPathDocument, and XmlDocument

The `XPathDocument` is so named because it is the most efficient way to use XPath expressions over an in-memory data structure. The `XPathDocument` implements the `IXPathNavigable` interface, allowing you to iterate over the underlying XML by providing an `XPathNavigator`. The `XPathNavigator` class differs from the `XmlReader` because rather than forward-only, it provides random access over your XML, similar to a read-only ADO Keyset recordset versus a forward-only recordset.

You typically want to use an `XPathDocument` to move around freely, forward and backward, within a document. `XPathDocument` is read-only, while `XmlDocument` allows read-write access.

The `XmlDocument` in version 2.0 adds in-memory validation. Using the `XmlReader`, the only way to validate the XML is from a stream or file. The `XmlDocument` now allows in-memory validation without the file or stream access using `Validate()`. `XmlDocument` also adds capability to subscribe to events like `NodeChanged`, `NodeInserting`, and the like.

XPath is a query language best learned by example. You must know it to make good use of the `XPathDocument`. Here are some valid XPath queries that you can use with the `Books.xml` file. XPath is a rich language in its own right, with many dozens of functions. As such, fully exploring XPath is beyond the scope of this book, but the following table should give you a taste of what's possible.

Xpath Function	Result
`//book[@genre = "novel"]/title`	Recursively from the root node, gets the titles of all books whose genre attribute is equal to `novel`
`/bookstore/book[author/last-name = "Melville"]`	Gets all books that are children of `bookstore` whose author's last name is `Melville`
`/bookstore/book/author[last-name = "Melville"]`	Gets all authors that are children of `book` whose last name is `Melville`
`//book[title = "The Gorgias" or title = "The Confidence Man"]`	Recursively from the root node, gets all books whose title is either `The Gorgias` or `The Confidence Man`
`//title[contains(., "The")]`	Gets all titles that contain the string `The`
`//book[not(price[.>10.00])]`	Gets all books whose prices are not greater than `10.00`

Listing 10-12 queries an `XPathDocument` for books whose prices are less than $10.00 and outputs the price. In order to illustrate using built-in XPath functions, this example uses a greater-than instead of using a less-than. It then inverts the result using the built-in `not()` method. XPath includes a number of functions for string concatenation, arithmetic, and many other uses. The `XPathDocument` returns an `XPathNavigator` as a result of calling `CreateNavigator`. The `XPathNavigator` is queried using an XPath passed to the `Select` method and returns an `XPathNodeIterator`. That `XPathNodeIterator` is `foreach` enabled via `IEnumerable`. As Listing 10-16 uses a read-only `XPathDocument`, it will not update the data in memory.

Listing 10-12: Querying XML with XPathDocument and XPathNodeIterator

VB

```
'Load document
Dim booksFile As String = Server.MapPath("books.xml")
Dim document As New XPathDocument(booksFile)
Dim nav As XPathNavigator = document.CreateNavigator()

'Add a namespace prefix that can be used in the XPath expression
Dim namespaceMgr As New XmlNamespaceManager(nav.NameTable)
```

```
namespaceMgr.AddNamespace("b", "http://example.books.com")

'All books whose price is not greater than 10.00
For Each node As XPathNavigator In nav.Select( _
        "//b:book[not(b:price[. > 10.00])]/b:price", namespaceMgr)
    Dim price As Decimal = _
        CType(node.ValueAs(GetType(Decimal)), Decimal)
    Response.Write(String.Format("Price is {0}<BR/>", _
        price))
Next
```

C#

```
//Load document
string booksFile = Server.MapPath("books.xml");

XPathDocument document = new XPathDocument(booksFile);
XPathNavigator nav = document.CreateNavigator();

//Add a namespace prefix that can be used in the XPath expression
XmlNamespaceManager namespaceMgr = new XmlNamespaceManager(nav.NameTable);
namespaceMgr.AddNamespace("b", "http://example.books.com");

//All books whose price is not greater than 10.00
foreach(XPathNavigator node in
    nav.Select("//b:book[not(b:price[. > 10.00])]/b:price",
    namespaceMgr))
{
    Decimal price = (decimal)node.ValueAs(typeof(decimal));
    Response.Write(String.Format("Price is {0}<BR/>",
        price));
}
```

If you then want to modify the underlying XML nodes, in the form of an `XPathNavigator`, you would use an XmlDocument instead of an XPathDocument. Your XPath expression evaluation may slow you down, but you will gain the capability to edit. Beware of this tradeoff in performance. Most often, you will want to use the read-only XPathDocument whenever possible. Listing 10-13 illustrates this change with the new or changed portions appearing in gray. Additionally, now that the document is editable, the price is increased 20 percent.

Listing 10-13: Querying and editing XML with XmlDocument and XPathNodeIterator

VB

```
'Load document
Dim booksFile As String = Server.MapPath("books.xml")

Dim document As New XmlDocument()
document.Load(booksFile)
Dim nav As XPathNavigator = document.CreateNavigator()

'Add a namespace prefix that can be used in the XPath expression
Dim namespaceMgr As New XmlNamespaceManager(nav.NameTable)
namespaceMgr.AddNamespace("b", "http://example.books.com")
'All books whose price is not greater than 10.00
For Each node As XPathNavigator In nav.Select( _
            "//b:book[not(b:price[. > 10.00])]/b:price", namespaceMgr)
        Dim price As Decimal = CType(node.ValueAs(GetType(Decimal)), Decimal)
        node.SetTypedValue(price * CDec(1.2))
        Response.Write(String.Format("Price raised from {0} to {1}<BR/>", _
            price, _
            CType(node.ValueAs(GetType(Decimal)), Decimal)))
    Next
```

C#

```
//Load document
string booksFile = Server.MapPath("books.xml");

XmlDocument document = new XmlDocument();
document.Load(booksFile);
XPathNavigator nav = document.CreateNavigator();

//Add a namespace prefix that can be used in the XPath expression
XmlNamespaceManager namespaceMgr = new XmlNamespaceManager(nav.NameTable);
namespaceMgr.AddNamespace("b", "http://example.books.com");

//All books whose price is not greater than 10.00
foreach(XPathNavigator node in
    nav.Select("//b:book[not(b:price[. > 10.00])]/b:price",
    namespaceMgr))
{
    Decimal price = (decimal)node.ValueAs(typeof(decimal));
    node.SetTypedValue(price * 1.2M);
    Response.Write(String.Format("Price inflated raised from {0} to {1}<BR/>",
        price,
        node.ValueAs(typeof(decimal))));
}
```

Listing 10-13 changes the `XPathDocument` to an `XmlDocument`, and adds a call to `XPathNavigator.SetTypedValue` to update the price of the document in memory. The resulting document could then be persisted to storage as needed. If `SetTypedValue` was instead called on the `XPathNavigator` that was returned by `XPathDocument`, a `NotSupportedException` would be thrown as the `XPathDocument` is read-only.

The `Books.xml` document loaded from disk uses `http://example.books.com` as its default namespace. Because the `Books.xsd` XML Schema is associated with the `Books.xml` document, and it assigns the default namespace to be `http://example.books.com`, the XPath must know how to resolve that namespace. Otherwise, you cannot determine if an XPath expression with the word `book` in it refers to a book from this namespace or another book entirely. An `XmlNamespaceManager` is created, and `b` is arbitrarily used as the namespace prefix for the XPath expression.

Namespace resolution can be very confusing because it is easy to assume that your XML file is all alone in the world and that specifying a node named `book` is specific enough to enable the system to find it. However, remember that your XML documents should be thought of as living among all the XML in the world — this makes providing a qualified namespace all the more important. The `XmlNamespaceManager` in Listing 10-12 is passed into the call to `SelectNodes` in order to associate the prefix with the appropriate namespace. Remember, the namespace is unique, not the prefix; the prefix is simply a convenience acting as an alias to the longer namespace. If you find that you're having trouble getting an XPath expression to work and no nodes are being returned, find out if your source XML has a namespace specified and that it matches up with a namespace in your XPath.

Using XPath with XDocuments in LINQ for XML

You can use XPath against an `XDocument` object by adding a reference to the `System.Xml.XPath` namespace via a using or Imports statement. Adding this reference adds new extension methods to the `XDocument` like `CreateNavigator` get to an `XPathNavigator` and the very useful XPathSelectElements. XPathSelectElements is similar to the `SelectNodes` and `SelectSingleNode` methods of the `System.Xml.XmlDocument`. These extension methods are part of the ''bridge classes'' that provide smooth integration between `System.Xml` and `System.Xml.Linq`.

Listing 10-12q: Querying XDocuments with XPath Expressions

VB

```
Dim booksFile As String = Server.MapPath("books.xml")
Dim document As XDocument = XDocument.Load(booksFile)

'Add a namespace prefix that can be used in the XPath expression
Dim namespaceMgr As New XmlNamespaceManager(New NameTable())
namespaceMgr.AddNamespace("b", "http://example.books.com")

'All books whose price is not greater than 10.00
Dim nodes = document.XPathSelectElements(
    "//b:book[not(b:price[. > 10.00])]/b:price", namespaceMgr)

For Each node In nodes
    Response.Write(node.Value + "<BR/>")
Next
```

C#

```
//Load document
string booksFile = Server.MapPath("books.xml");
XDocument document = XDocument.Load(booksFile);

//Add a namespace prefix that can be used in the XPath expression.
// Note the need for a NameTable. It could be new or come from elsewhere.
XmlNamespaceManager namespaceMgr = new XmlNamespaceManager(new NameTable());
namespaceMgr.AddNamespace("b", "http://example.books.com");

var nodes = document.XPathSelectElements(
    "//b:book[not(b:price[. > 10.00])]/b:price",namespaceMgr);

//All books whose price is not greater than 10.00
foreach (var node in nodes)
{
    Response.Write(node.Value + "<BR/>");
}
```

Notice that the added method in Listing 10-12q, `XPathSelectElements`, still requires an `IXmlNamespaceResolver`, so we create a new `NameTable` and map the namespaces and prefixes explicitly via `XmlNamespaceManager`. When using XElements and simple queries, you're better off using LINQ to XML and the new XElement-specific methods such as `Elements()` and `Descendants()` rather than XPath.

DataSets

The `System.Data` namespace and `System.Xml` namespace have started mingling their functionality for some time. DataSets are a good example of how relational data and XML data meet in a hybrid class library. During the COM and XML heyday, the ADO 2.5 recordset sported the capability to persist as XML. The dramatic inclusion of XML functionality in a class library focused entirely on manipulation of relational data was a boon for developer productivity. XML could be pulled out of SQL Server and manipulated.

Persisting DataSets to XML

Classes within `System.Data` use `XmlWriter` and `XmlReader` in a number of places. Now that you're more familiar with `System.Xml` concepts, be sure to take note of the method overloads provided by the classes within `System.Data`. For example, the `DataSet.WriteXml` method has four overloads, one of which takes in `XmlWriter`. Most of the methods with `System.Data` are very pluggable with the classes from `System.Xml`. Listing 10-14 shows another way to retrieve the XML from relational data by loading a DataSet from a SQL command and writing it directly to the browser with the `Response` object's `OutputStream` property using `DataSet.WriteXml`.

Listing 10-14: Extracting XML from a SQL Server with System.Data.DataSet

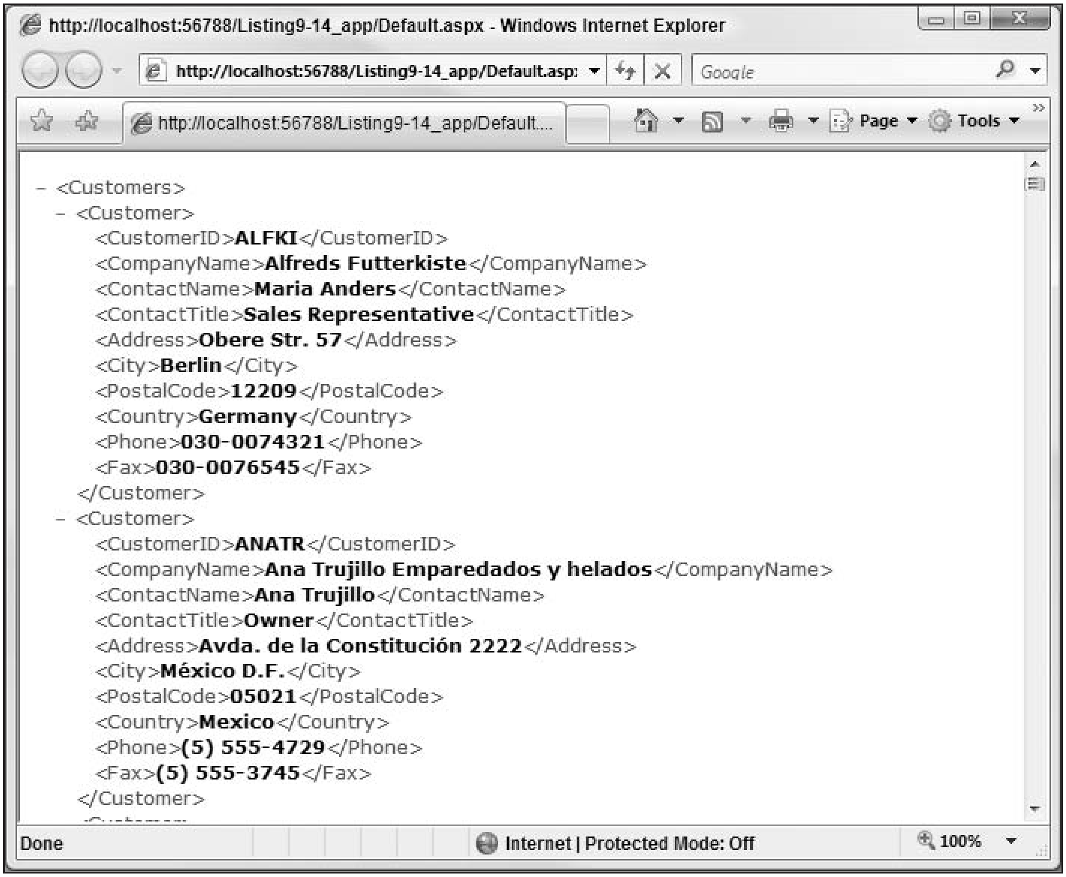
VB

```
Dim connStr As String = "database=Northwind;Data Source=localhost; " _
        & "User id=sa;pwd=wrox"

Using conn As New SqlConnection(connStr)
    Dim command As New SqlCommand("select * from customers", conn)
    conn.Open()
    Dim ds As New DataSet()
    ds.DataSetName = "Customers"
    ds.Load(command.ExecuteReader(), LoadOption.OverwriteChanges, "Customer")
    Response.ContentType = "text/xml"
    ds.WriteXml(Response.OutputStream)
End Using
```

C#

```
string connStr = "database=Northwind;Data Source=localhost;User id=sa;pwd=wrox";

using (SqlConnection conn = new SqlConnection(connStr))
{
    SqlCommand command = new SqlCommand("select * from customers", conn);
    conn.Open();
    DataSet ds = new DataSet();
    ds.DataSetName = "Customers";
    ds.Load(command.ExecuteReader(), LoadOption.OverwriteChanges, "Customer");
    Response.ContentType = "text/xml";
    ds.WriteXml(Response.OutputStream);
}
```

DataSets have a fairly fixed format, as illustrated in this example. The root node of the document is `Customers`, which corresponds to the `DataSetName` property. DataSets contain one or more named `DataTable` objects, and the names of these `DataTables` define the wrapper element — in this case, `Customer`. The name of the `DataTable` is passed into the `load` method of the DataSet. The correlation between the DataSet's name, `DataTable`'s name, and the resulting XML is not obvious when using DataSets. The resulting XML is shown in the browser in Figure 10-4.


http://localhost:56788/Listing9-14_app/Default.aspx - Windows Internet Explorer
http://localhost:56788/Listing9-14_app/Default.asp:
Google
http://localhost:56788/Listing9-14_app/Default....
Page
Tools
Done
Internet | Protected Mode: Off
100%

```
- <Customers>
  - <Customer>
      <CustomerID>ALFKI</CustomerID>
      <CompanyName>Alfreds Futterkiste</CompanyName>
      <ContactName>Maria Anders</ContactName>
      <ContactTitle>Sales Representative</ContactTitle>
      <Address>Obere Str. 57</Address>
      <City>Berlin</City>
      <PostalCode>12209</PostalCode>
      <Country>Germany</Country>
      <Phone>030-0074321</Phone>
      <Fax>030-0076545</Fax>
    </Customer>
  - <Customer>
      <CustomerID>ANATR</CustomerID>
      <CompanyName>Ana Trujillo Emparedados y helados</CompanyName>
      <ContactName>Ana Trujillo</ContactName>
      <ContactTitle>Owner</ContactTitle>
      <Address>Avda. de la Constitución 2222</Address>
      <City>México D.F.</City>
      <PostalCode>05021</PostalCode>
      <Country>Mexico</Country>
      <Phone>(5) 555-4729</Phone>
      <Fax>(5) 555-3745</Fax>
    </Customer>
```

Figure 10-4

DataSets present a data model that is very different from the XML way of thinking about data. Much of the XML-style of thinking revolves around the InfoSet or the DOM, whereas DataSets are row- and column-based. The `XmlDataDocument` is an attempt to present these two ways of thinking into one relatively unified model.

XmlDataDocument

Although DataSets have their own relatively inflexible format for using XML, the `XmlDocument` class does not. In order to bridge this gap, an unusual hybrid object, the `XmlDataDocument`, is introduced. This object maintains the full fidelity of all the XML structure and allows you to access XML via the `XmlDocument` API without losing the flexibility of a relational API. An `XmlDataDocument` contains a DataSet of its own and can be called DataSet-aware. Its internal DataSet offers a relational view of the XML data. Any data contained within the XML data document that does not map into the relational view is not lost, but becomes available to the DataSet's APIs.

The `XmlDataDocument` is a constructor that takes a DataSet as a parameter. Any changes made to the `XmlDataDocument` are reflected in the DataSet and vice versa.

Now take the DataSet loaded in Listing 10-14 and manipulate the data with the `XmlDataDocument` and DOM APIs you're familiar with. Next, jump back into the world of `System.Data` and see that the DataSets underlying DataRows have been updated with the new data, as shown in Listing 10-15.

Listing 10-15: Changing DataSets using the DOM APIs from XmlDataDocument

VB

```
Dim connStr As String = "database=Northwind;Data Source=localhost; " _
        & "User id=sa;pwd=wrox"

Using conn As New SqlConnection(connStr)
    Dim command As New SqlCommand("select * from customers", conn)
    conn.Open()
    Dim ds As New DataSet()
    ds.DataSetName = "Customers"
    ds.Load(command.ExecuteReader(), LoadOption.OverwriteChanges, "Customer")
    'Response.ContentType = "text/xml"
    'ds.WriteXml(Response.OutputStream)

    'Added in Listing 10-15
    Dim doc As New XmlDataDocument(ds)
    doc.DataSet.EnforceConstraints = False
    Dim node As XmlNode = _
    doc.SelectSingleNode("//Customer[CustomerID = 'ANATR']/ContactTitle")
    node.InnerText = "Boss"
    doc.DataSet.EnforceConstraints = True

    Dim dr As DataRow = doc.GetRowFromElement(CType(node.ParentNode, XmlElement))
    Response.Write(dr("ContactName").ToString() & " is the ")
    Response.Write(dr("ContactTitle").ToString())
End Using
```

C#

```
string connStr = "database=Northwind;Data Source=localhost; "
        + "User id=sa;pwd=wrox";

using (SqlConnection conn = new SqlConnection(connStr))
{
    SqlCommand command = new SqlCommand("select * from customers", conn);
    conn.Open();
    DataSet ds = new DataSet();
    ds.DataSetName = "Customers";
    ds.Load(command.ExecuteReader(), LoadOption.OverwriteChanges,"Customer");
    //Response.ContentType = "text/xml";
    //ds.WriteXml(Response.OutputStream);

    //Added in Listing 10-15
    XmlDataDocument doc = new XmlDataDocument(ds);
    doc.DataSet.EnforceConstraints = false;
    XmlNode node = doc.SelectSingleNode(@"//Customer[CustomerID
        = 'ANATR']/ContactTitle");
        node.InnerText = "Boss";
        doc.DataSet.EnforceConstraints = true;

        DataRow dr = doc.GetRowFromElement((XmlElement)node.ParentNode);
        Response.Write(dr["ContactName"].ToString() + " is the ");
        Response.Write(dr["ContactTitle"].ToString());
    }
```

Listing 10-15 extends Listing 10-14 by first commenting out changing the HTTP `ContentType` and the call to `DataSet.WriteXml`. After the DataSet is loaded from the database, it is passed to the `XmlDataDocument` constructor. At this point, the `XmlDataDocument` and the DataSet refer to the same set of information. The `EnforceConstraints` property of the DataSet is set to `false` to allow changes to the DataSet. When `EnforceConstraints` is later set to `true`, if any constraint rules were broken, an exception is thrown. An `XPath` expression is passed to the DOM method `SelectSingleNode`, selecting the `ContactTitle` node of a particular customer, and its text is changed to `Boss`. Then by calling `GetRowFromElement` on the `XmlDataDocument`, the context jumps from the world of the `XmlDocument` back to the world of the DataSet. Column names are passed into the `indexing` property of the returned `DataRow`, and the output is shown in this line:

```
Ana Trujillo is the Boss
```

The data is loaded from the SQL server and then manipulated and edited with `XmlDocument`-style methods; a string is then built using a `DataRow` from the underlying DataSet.

XML is clearly more than just angle brackets. XML data can come from files, from databases, from information sets like the DataSet object, and certainly from the Web. Today, however, a considerable amount of data is stored in XML format, so a specific data source control has been added to ASP.NET 2.0 just for retrieving and working with XML data.

The XmlDataSource Control

The `XmlDataSource` control enables you to connect to your XML data and to use this data with any of the ASP.NET data-bound controls. Just like the `SqlDataSource` and the `AccessDataSource` controls, the `XmlDataSource` control also enables you not only to retrieve data, but also to insert, delete, and update data items.

One unfortunate caveat of the `XmlDataSource` is that its `XPath` attribute does not support documents that use namespace qualification. Examples in this chapter use the `Books.xml` file with a default namespace of `http://examples.books.com`. It is very common for XML files to use multiple namespaces, including a default namespace. As you learned when you created an `XPathDocument` and queried it with XPath, the namespace in which an element exists is very important. The regrettable reality is, there is no way to use a namespace qualified XPath expression or to make the XmlDataSource Control aware of a list of prefix/namespace pairs via the `XmlNamespaceManager` class. However, the `XPath` function used in the `ItemTemplate` of the templated `DataList` control *can* take a `XmlNamespaceManager` as its second parameter and query XML returned from the `XmlDataSource` — as long as the control does not include an `XPath` attribute with namespace qualification or you can

just omit it all together. That said, in order for these examples to work, you must remove the namespaces from your source XML and use XPath queries that include no namespace qualification, as shown in Listing 10-16.

You can use a `DataList` control or any DataBinding-aware control and connect to an `<asp: XmlDataSource>` control. The technique for binding a control directly to the `Books.xml` file is illustrated in Listing 10-16.

Listing 10-16: Using a DataList control to display XML content

```
<%@ Page Language="VB" AutoEventWireup="false"
CodeFile="Default.aspx.vb" Inherits="Default_aspx" %>
<html xmlns="http://www.w3.org/1999/xhtml" >
    <head id="Head1" runat="server">
        <title>XmlDataSource</title>
    </head>
    <body>
    <form id="form1" runat="server">
        <asp:datalist id="DataList1" DataSourceID="XmlDataSource1" runat="server">
            <ItemTemplate>
                <p><b><%# XPath("author/first-name") %>
                    <%# XPath("author/last-name")%></b>
                    wrote <%# XPath("title") %></p>
            </ItemTemplate>
        </asp:datalist>
        <asp:xmldatasource id="XmlDataSource1" runat="server"
            datafile="~/Books.xml"
            xpath="//bookstore/book"/>
    </form>
    </body>
</html>
```

This is a simple example, but it shows you the ease of using the `XmlDataSource` control. You should focus on two attributes in this example. The first is the `DataFile` attribute. This attribute points to the location of the XML file. Because the file resides in the root directory of the application, it is simply `~/Books.xml`. The next attribute included in the `XmlDataSource` control is the `XPath` attribute. The `XmlDataSource` control uses the `XPath` attribute for the filtering of XML data. In this case, the `XmlDataSource` control is taking everything within the `<book>` set of elements. The value `//bookstore/book` means that the `XmlDataSource` control navigates to the `<bookstore>` element and then to the `<book>` element within the specified XML file and returns a list of all books.

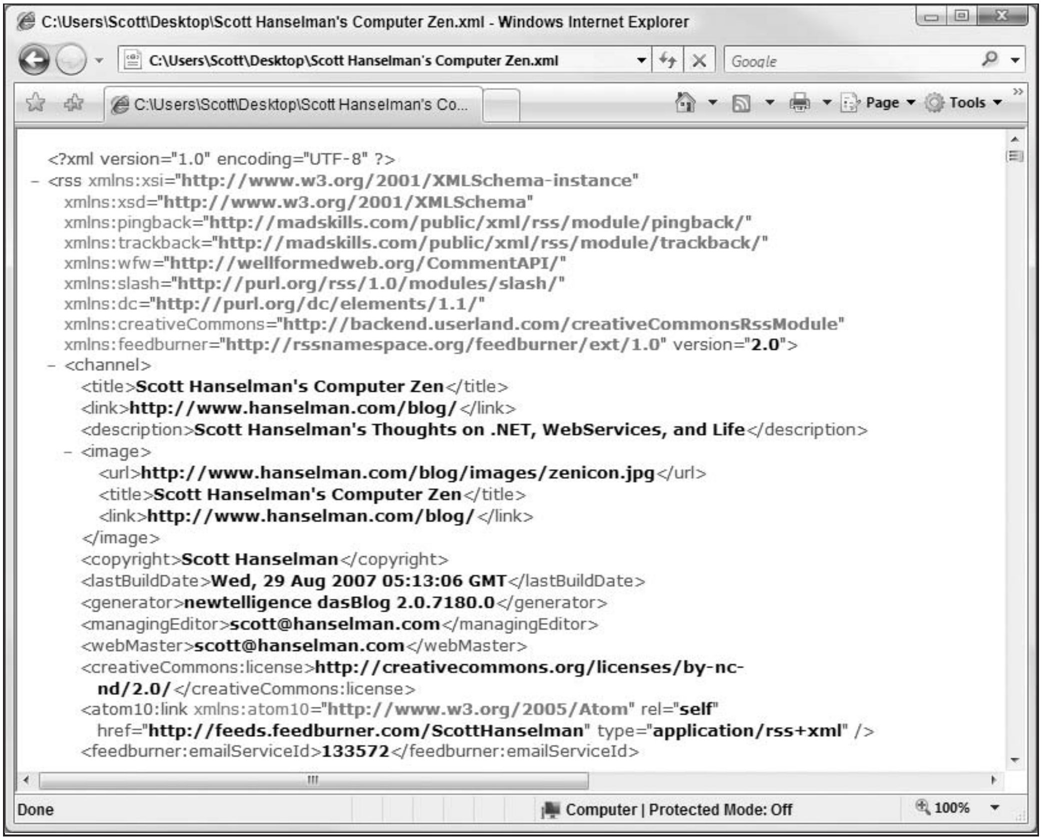
The `DataList` control then must specify its `DataSourceID` as the `XmlDataSource` control. In the `<ItemTemplate>` section of the `DataList` control, you can retrieve specific values from the XML file by using XPath commands within the template. The XPath commands filter the data from the XML file. The first value retrieved is an element attribute (`author/first-name`) that is contained in the `<book>` element. If you are retrieving an attribute of an element, you preface the name of the attribute with an *at* (`@`) symbol. The next two XPath commands get the last name and the title of the book. Remember to separate nodes with a forward slash (`/`). When run in the browser, this code produces the results illustrated in the following list:

```
Benjamin Franklin wrote The Autobiography of Benjamin Franklin
Herman Melville wrote The Confidence Man
Sidas Plato wrote The Gorgias
```

Note that if you wrote the actual code, this entire exercise would be done entirely in the ASPX page itself!

Besides working from static XML files such as the `Books.xml` file shown earlier, the `XmlDataSource` control has the capability to work from dynamic, URL-accessible XML files. One popular XML format that is pervasive on the Internet today is the *weblog*. These *blogs*, or personal diaries, can be viewed either in the browser, through an RSS-aggregator, or as pure XML.

In Figure 10-5, you can see the XML from my blog's RSS feed. I've saved the XML to a local file and removed a stylesheet so I can see what the XML looks like when viewed directly in the browser. (You can find a lot of blogs to play with for this example at `weblogs.asp.net`.)


C:\Users\Scott\Desktop\Scott Hanselman's Computer Zen.xml - Windows Internet Explorer
C:\Users\Scott\Desktop\Scott Hanselman's Computer Zen.xml
Google
C:\Users\Scott\Desktop\Scott Hanselman's Co...
Page
Tools
Done
Computer | Protected Mode: Off
100%

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <rss xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:pingback="http://madskills.com/public/xml/rss/module/pingback/"
    xmlns:trackback="http://madskills.com/public/xml/rss/module/trackback/"
    xmlns:wfw="http://wellformedweb.org/CommentAPI/"
    xmlns:slash="http://purl.org/rss/1.0/modules/slash/"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:creativeCommons="http://backend.userland.com/creativeCommonsRssModule"
    xmlns:feedburner="http://rssnamespace.org/feedburner/ext/1.0" version="2.0">
  - <channel>
      <title>Scott Hanselman's Computer Zen</title>
      <link>http://www.hanselman.com/blog/</link>
      <description>Scott Hanselman's Thoughts on .NET, WebServices, and Life</description>
    - <image>
        <url>http://www.hanselman.com/blog/images/zenicon.jpg</url>
        <title>Scott Hanselman's Computer Zen</title>
        <link>http://www.hanselman.com/blog/</link>
      </image>
      <copyright>Scott Hanselman</copyright>
      <lastBuildDate>Wed, 29 Aug 2007 05:13:06 GMT</lastBuildDate>
      <generator>newtelligence dasBlog 2.0.7180.0</generator>
      <managingEditor>scott@hanselman.com</managingEditor>
      <webMaster>scott@hanselman.com</webMaster>
      <creativeCommons:license>http://creativecommons.org/licenses/by-nc-
        nd/2.0/</creativeCommons:license>
      <atom10:link xmlns:atom10="http://www.w3.org/2005/Atom" rel="self"
        href="http://feeds.feedburner.com/ScottHanselman" type="application/rss+xml" />
      <feedburner:emailServiceId>133572</feedburner:emailServiceId>
```

Figure 10-5

Now that you know the location of the XML from the blog, you can use this XML with the `XmlDataSource` control and display some of the results in a `DataList` control. The code for this example is shown in Listing 10-17.

Listing 10-17: Displaying an XML RSS blog feed

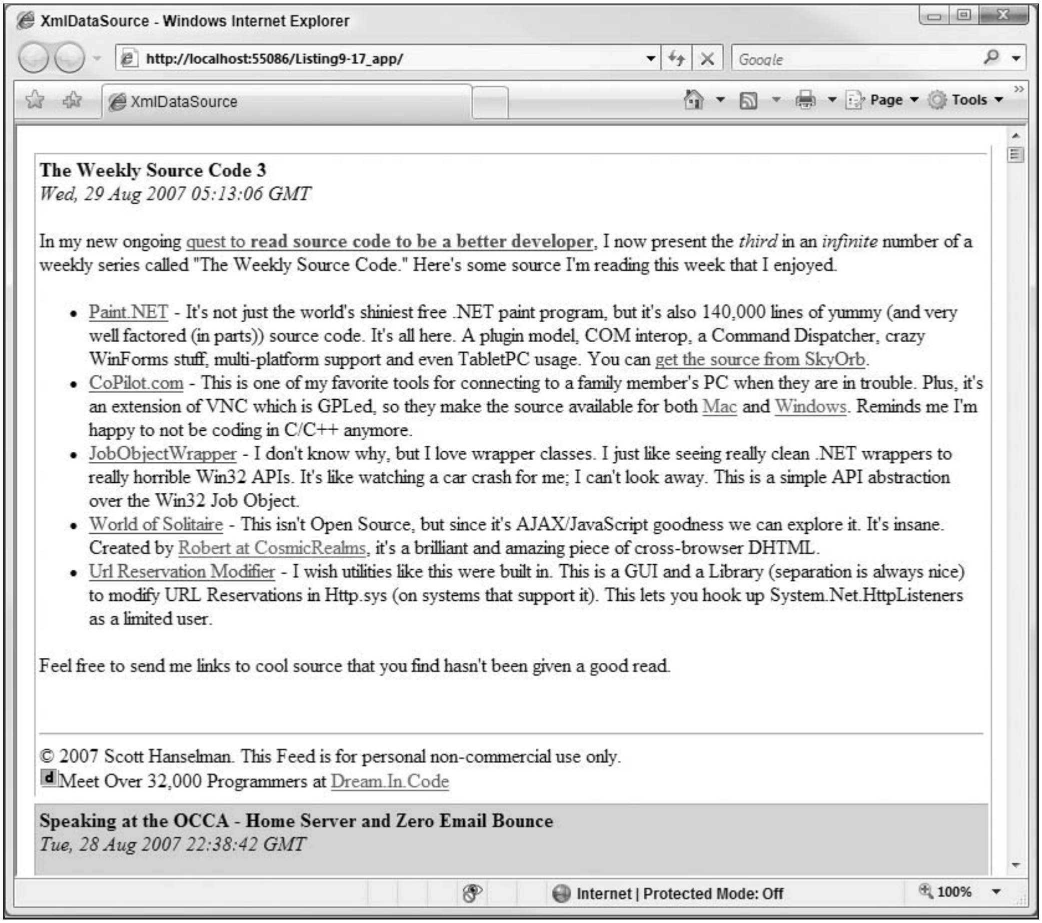
```
<%@ Page Language="VB"%>
<html xmlns="http://www.w3.org/1999/xhtml" >
    <head runat="server">
        <title>XmlDataSource</title>
    </head>
    <body>
    <form id="form1" runat="server">
        <asp:DataList ID="DataList1" Runat="server" DataSourceID="XmlDataSource1">
            <HeaderTemplate>
                <table border="1" cellpadding="3">
            </HeaderTemplate>
            <ItemTemplate>
                <tr><td><b><%# XPath("title") %></b><br />
                <i><%# XPath("pubDate") %></i><br />
                <%# XPath("description") %></td></tr>
            </ItemTemplate>
            <AlternatingItemTemplate>
                <tr bgcolor="LightGrey"><td><b><%# XPath("title") %></b><br />
                <i><%# XPath("pubDate") %></i><br />
                <%# XPath("description") %></td></tr>
            </AlternatingItemTemplate>
            <FooterTemplate>
                </table>
            </FooterTemplate>
        </asp:DataList>
        <asp:XmlDataSource ID="XmlDataSource1" Runat="server"
        DataFile="http://www.hanselman.com/blog/feed"
        XPath="rss/channel/item">
        </asp:XmlDataSource>
    </form>
    </body>
</html>
```

Looking at the code in Listing 10-17, you can see that the `DataFile` points to a URL where the XML is retrieved. The `XPath` property filters and returns all the `<item>` elements from the RSS feed. The `DataList` control creates an HTML table and pulls out specific data elements from the RSS feed, such as the `<title>`, `<pubDate>`, and `<description>` elements.

Running this page in the browser, you get something similar to the results shown in Figure 10-6.

This approach also works with XML Web services, even ones for which you can pass in parameters using HTTP-GET. You just set up the `DataFile` property value in the following manner:

```
DataFile="http://www.someserver.com/GetWeather.asmx/ZipWeather?zipcode=63301"
```


XmlDataSource - Windows Internet Explorer
http://localhost:55086/Listing9-17_app/
Google
XmlDataSource
Page
Tools
Internet | Protected Mode: Off
100%

The Weekly Source Code 3
Wed, 29 Aug 2007 05:13:06 GMT

In my new ongoing quest to **read source code to be a better developer**, I now present the *third* in an *infinite* number of a weekly series called "The Weekly Source Code." Here's some source I'm reading this week that I enjoyed.

- Paint.NET - It's not just the world's shiniest free .NET paint program, but it's also 140,000 lines of yummy (and very well factored (in parts)) source code. It's all here. A plugin model, COM interop, a Command Dispatcher, crazy WinForms stuff, multi-platform support and even TabletPC usage. You can get the source from SkyOrb.
- CoPilot.com - This is one of my favorite tools for connecting to a family member's PC when they are in trouble. Plus, it's an extension of VNC which is GPLed, so they make the source available for both Mac and Windows. Reminds me I'm happy to not be coding in C/C++ anymore.
- JobObjectWrapper - I don't know why, but I love wrapper classes. I just like seeing really clean .NET wrappers to really horrible Win32 APIs. It's like watching a car crash for me; I can't look away. This is a simple API abstraction over the Win32 Job Object.
- World of Solitaire - This isn't Open Source, but since it's AJAX/JavaScript goodness we can explore it. It's insane. Created by Robert at CosmicRealms, it's a brilliant and amazing piece of cross-browser DHTML.
- Url Reservation Modifier - I wish utilities like this were built in. This is a GUI and a Library (separation is always nice) to modify URL Reservations in Http.sys (on systems that support it). This lets you hook up System.Net.HttpListeners as a limited user.

Feel free to send me links to cool source that you find hasn't been given a good read.

© 2007 Scott Hanselman. This Feed is for personal non-commercial use only.
d Meet Over 32,000 Programmers at Dream.In.Code

Speaking at the OCCA - Home Server and Zero Email Bounce
Tue, 28 Aug 2007 22:38:42 GMT

Figure 10-6

There is no end to the number of places you can find and use XML: files, databases, Web sites, and services. Sometimes you will want to manipulate the XML via queries or programmatically, and sometimes you will want to take the XML ‘‘tree’’ and transform it into a tree of a different form.

XSLT

XSLT is a tree transformation language also written in XML syntax. It’s a strange hybrid of a declarative and a programmatic language, and some programmers would argue that it’s not a language at all. Others, who use a number of XSLT scripting extensions, would argue that it is a very powerful language. Regardless of the controversy, XSLT transformations are very useful for changing the structure of XML files quickly and easily, often using a very declarative syntax.

The best way to familiarize yourself with XSLT is to look at an example. Remember that the `Books.xml` file used in this chapter is a list of books and their authors. The XSLT in Listing 10-18 takes that document and transforms it into a document that is a list of authors.

Listing 10-18: Books.xslt

XSLT

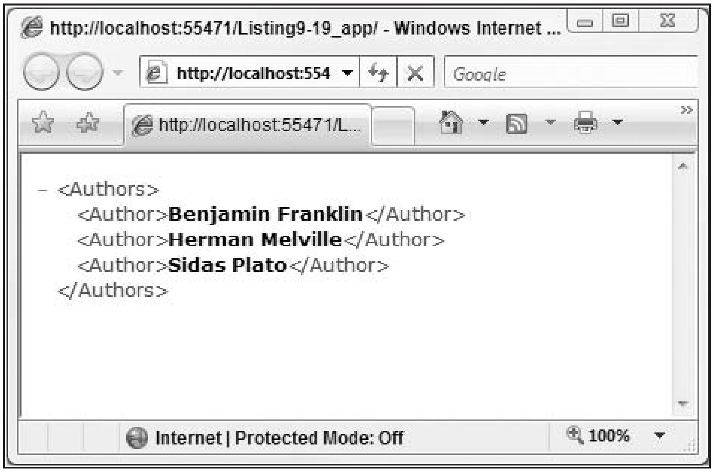
```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:template match="/">
        <xsl:element name="Authors">
            <xsl:apply-templates select="//book"/>
        </xsl:element>
    </xsl:template>
    <xsl:template match="book">
        <xsl:element name="Author">
            <xsl:value-of select="author/first-name"/>
            <xsl:text> </xsl:text>
            <xsl:value-of select="author/last-name"/>
        </xsl:element>
    </xsl:template>
</xsl:stylesheet>
```

Remember that XSLT is XML vocabulary in its own right, so it makes sense that it has its own namespace and namespace prefix. XSLT is typically structured with a series of templates that match elements in the source document. The XSLT document doesn't describe what the result looks like as much as it declares what steps must occur for the transformation to succeed. Remembering that your goal is an XML file with a list of authors, you match on the root node of `Books.xml` and output a root element for the resulting document named `<Authors>`. Then `<xsl:apply-templates select = "//book"/>` indicates to the processor that it should continue looking for templates that, in this case, match the XPath expression `//book`. Below the first template is a second template that handles all book matches. It outputs a new element named `<Author>`.

XSLT is very focused on context, so it is often helpful to imagine a cursor that is on a particular element of the source document. Immediately after outputting the `<Author>` element, the processor is in the middle of the template match on the `book` element. All XPath expressions in this example are relative to the `book` element. So the `<xsl:value-of select = "author/first-name">` directive searches for the author's first name relative to the `book` element. The `<xsl:text> </xsl:text>` directive is interesting to note because it is explicit and a reminder that a difference exists between significant white space and insignificant white space. It is important, for example, that a space is put between the author's first and last names, so it must be called out explicitly.

The resulting document is shown in Figure 10-7.

This example only scratches the surface of XSLT's power. Although a full exploration of XSLT is beyond the scope of this book, other books by Wrox Press cover the topic more fully. Remember that the .NET Framework implements the 1.0 implementation of XSLT. As of this writing, XSLT 2.0 and XPath 2.0 are W3C Recommendations, and Microsoft is working on CTPs (Community Technology Previews) of XSLT 2 functionality. More details are available on the Microsoft XmlTeam blog at `http://blogs.msdn.com/xmlteam/archive/2007/01/29/xslt-2-0.aspx`.


http://localhost:55471/Listing9-19_app/ - Windows Internet ...
http://localhost:554
Google
http://localhost:55471/L...
Internet | Protected Mode: Off
100%

```
- <Authors>
    <Author>Benjamin Franklin</Author>
    <Author>Herman Melville</Author>
    <Author>Sidas Plato</Author>
  </Authors>
```

Figure 10-7

Figure 10-7 shows the resulting XML as the `Books.xslt` transformation is applied to `Books.xml`. You can apply XSLT transformations in a number of ways, both declarative and programmatic. These are described in the following sections.

XslCompiledTransform

The `XslTransform` class was used in the .NET Framework 1.*x* for XSLT transformation. In the .NET Framework 2.0, the `XslCompiledTransform` class is the new XSLT processor. It is such an improvement that `XslTransform` is deprecated and marked with the `Obsolete` attribute. The compiler will now advise you to use `XslCompiledTransform`. The system generates MSIL code on the call to `Compile()` and the XSLT executes many times faster than previous techniques. This compilation technique also includes full debugging support from within Visual Studio, which is covered a little later in this chapter.

The `XPathDocument` is absolutely optimized for XSLT transformations and should be used instead of the `XmlDocument` if you would like a 15 to 30 percent performance gain in your transformations. Remember that XSLT contains XPath, and when you use XPath, use an `XPathDocument`. According to the team's numbers, XSLT is 400 percent faster in .NET Framework 2.0.

`XslCompiledTransform` has only two methods: `Load` and `Transform`. The compilation happens without any effort on your part. Listing 10-19 loads the `Books.xml` file into an `XPathDocument` and transforms it using `Books.xslt` and an `XslCompiledTransform`. Even though there are only two methods, there are *14* overrides for `Transform` and 6 for `Load`. That may seem a little daunting at first, but there is a simple explanation.

The `Load` method can handle loading a stylesheet from a string, an `XmlReader`, or any class that implements `IXPathNavigable`. An `XsltSettings` object can be passed in optionally with any of the previous three overloads, giving you six to choose from. `XsltSettings` includes options to enable the `document()` XSLT–specific function via the `XsltSettings.EnableDocumentFunction` property or enable embedded script blocks within XSLT via `XsltSettings.EnableScript`. These advanced options are disabled by default for security reasons. Alternatively, you can retrieve a pre-populated `XslSettings` object via the static property `XsltSettings.TrustedXslt`, which has enabled both these settings.

> **If you think it is odd that the class that does the work is called the `XslCompiledTransform` and not the `XsltCompiledTransform`, but `XsltSettings` includes the *t*, remember that the *t* in XSLT means *transformation*.**

Note in Listing 10-19 that the `Response.Output` property eliminates an unnecessary string allocation. In the example, `Response.Output` is a `TextWriter` wrapped in an `XmlTextWriter` and passed directly to the `Execute` method.

Listing 10-19: Executing an XsltCompiledTransform

VB

```
Response.ContentType = "text/xml"

Dim xsltFile As String = Server.MapPath("books.xslt")
Dim xmlFile As String = Server.MapPath("books.xml")

Dim xslt As New XslCompiledTransform()
xslt.Load(xsltFile)

Dim doc As New XPathDocument(xmlFile)
xslt.Transform(doc, New XmlTextWriter(Response.Output))
```

C#

```
Response.ContentType = "text/xml";

string xsltFile = Server.MapPath("books.xslt");
string xmlFile = Server.MapPath("books.xml");

XslCompiledTransform xslt = new XslCompiledTransform();
xslt.Load(xsltFile);

XPathDocument doc = new XPathDocument(xmlFile);
xslt.Transform(doc, new XmlTextWriter(Response.Output));
```

Named arguments may be passed into an `XslTransform` or `XslCompiledTransform` if the stylesheet takes parameters. The following code snippet illustrates the use of `XslArgumentList`:

```
XslTransform transformer = new XslTransform();
transformer.Load("foo.xslt");

XslArgumentList args = new XslArgumentList();
args.Add("ID", "SOMEVALUE");

transformer.Transform("foo.xml", args, Response.OutputStream);
```

The XML resulting from an XSLT transformation can be manipulated with any of the `system.XML` APIs that have been discussed in this chapter. One common use of XSLT is to flatten hierarchical and, sometimes, relational XML documents into a format that is more conducive to output as HTML. The results of these transformations to HTML can be placed inline within an existing ASPX document.

The new XSLTC.exe Command-Line Compiler

Compiled stylesheets are very useful but there is a slight performance hit as the stylesheets are compiled at runtime, so the .NET Framework 3.5 has introduced `XSLTC.exe`, a command-line XSLT compiler.

Usage is simple — you simply pass in as many source XSLT files as you like and specify the assembly output file. From a Visual Studio 2008 command prompt, use the following:

```
Xsltc /c:Wrox.Book.CompiledStylesheet books.xslt /out:Books.dll
```

Now, add a reference to the newly created `Books.dll` in your project from Listing 10-19, and change one line:

```
XslCompiledTransform xslt = new XslCompiledTransform();
xslt.Load(typeof(Wrox.Book.MyCompiledStyleSheet));
```

Rather than loading the XSLT from a file, it's now loaded pre-compiled directly from the generated assembly. Using the XSLT Compiler makes deployment much easier as you can put many XSLTs in a single assembly, but most important, you eliminate code-generation time.

If your XSLT uses msxsl:script elements, that code will be compiled into separate assemblies, one per language used. You can merge these resulting assemblies with ILMerge, located at `http://research.microsoft.com/~mbarnett/ILMerge.aspx`, as a post-build step.XML Web Server Control.

XSLT transformations can also be a very quick way to get information out to the browser as HTML. Consider this technique as yet another tool in your toolbox. HTML is a tree, and HTML is a cousin of XML, so an XML tree can be transformed into an HTML tree. A benefit of using XSLT transformations to create large amounts of static text, like HTML tables, is that the XSLT file can be kept external to the application. You can make quick changes to its formatting without a recompile. A problem when using XSLT transformations is that they can become large and very unruly when someone attempts to use them to generate the entire user interface experience. The practice was in vogue in the mid-nineties to use XSLT transformations to generate entire Web sites, but the usefulness of this technique breaks down when complex user interactions are introduced. That said, XSLT has a place, not only for transforming data from one format to another, but also for creating reasonable chunks of your user interface-as long as you don't go overboard.

In the next example, the output of the XSLT is HTML rather than XML. Note the use of the `<xsl:output method="html">` directive. When this directive is omitted, the default output of an XSLT transformation is XML. This template begins with a match on the root node. It is creating an HTML fragment rather than an entire HTML document. Its first output is the `<h3>` tag with some static text. Next comes a table tag and the header row, and then the `<xsl:apply-template>` element selects all books within the source XML document. For every `book` element in the source document, the second template is invoked with the responsibility of outputting one table row per book. Calls to `<xsl:value-of>` select each of the book's subnodes and output them within the `<td>` tags (see Listing 10-20).

Listing 10-20: BookstoHTML.xslt used with the XML Web Server Control

XSLT

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:b="http://example.books.com" version="1.0">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <h3>List of Authors</h3>
        <table border="1">
            <tr>
                <th>First</th><th>Last</th>
            </tr>
            <xsl:apply-templates select="//b:book"/>
        </table>
    </xsl:template>
    <xsl:template match="b:book">
        <tr>
            <td><xsl:value-of select="b:author/b:first-name"/></td>
            <td><xsl:value-of select="b:author/b:last-name"/></td>
        </tr>
    </xsl:template>
</xsl:stylesheet>
```

ASPX

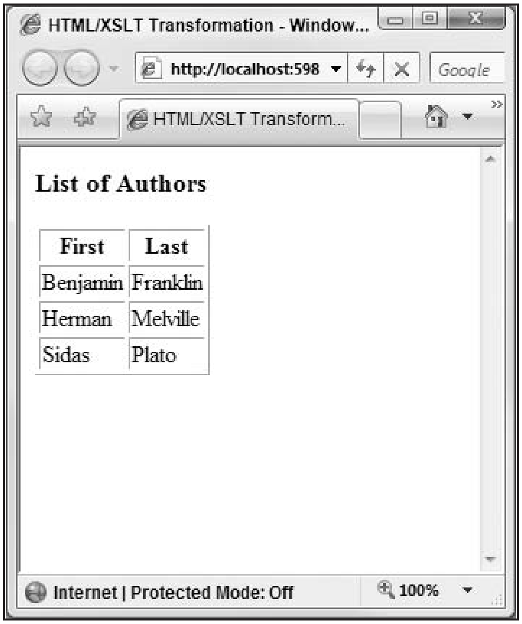
```
<%@ Page Language="VB" %>
<html xmlns="http://www.w3.org/1999/xhtml" >
    <head runat="server"><title>HTML/XSLT Transformation</title></head>
    <body>
        <form id="form1" runat="server">
            <div>
                <asp:Xml ID="Xml1" Runat="server"
                    DocumentSource="~/Books.xml"
                    TransformSource="~/bookstoHTML.xslt"/>
            </div>
        </form>
    </body>
</html>
```

Notice the use of namespace prefixes in Listing 10-20. The source namespace is declared with the prefix `b` as in `xmlns:b="http://example.books.com"` and the `b` prefix is subsequently used in XPath expressions like `//b:book`. The XSLT in Listing 10-20 can use the `XSLTCommand` to perform this transformation on the server-side because the entire operation is declarative and requires just two inputs — the XML document and the XSLT document. The XML Web server control makes the transformation easy to perform from the ASPX page and does not require any language-specific features. The `DocumentSource` property of the control holds the path to the `Books.xml` file, whereas the `TransformSource` property holds the path to the `BookstoHTML.xslt` file:

```
<h3>List of Authors</h3>
<table border="1">
    <tr>
        <th>First</th>
        <th>Last</th>
    </tr>
```

```
        <tr>
            <td>Benjamin</td>
            <td>Franklin</td>
        </tr>
        <tr>
            <td>Herman</td>
            <td>Melville</td>
        </tr>
        <tr>
            <td>Sidas</td>
            <td>Plato</td>
        </tr>
    </table>
```

The results of this transformation are output inline to this HTML document and appear between the two `<div>` tags. You see the results of this HTML fragment in the previous code and in the browser's output shown in Figure 10-8.


HTML/XSLT Transformation - Window...
http://localhost:598
Google
HTML/XSLT Transform...
Internet | Protected Mode: Off
100%

List of Authors

First	Last
Benjamin	Franklin
Herman	Melville
Sidas	Plato

Figure 10-8

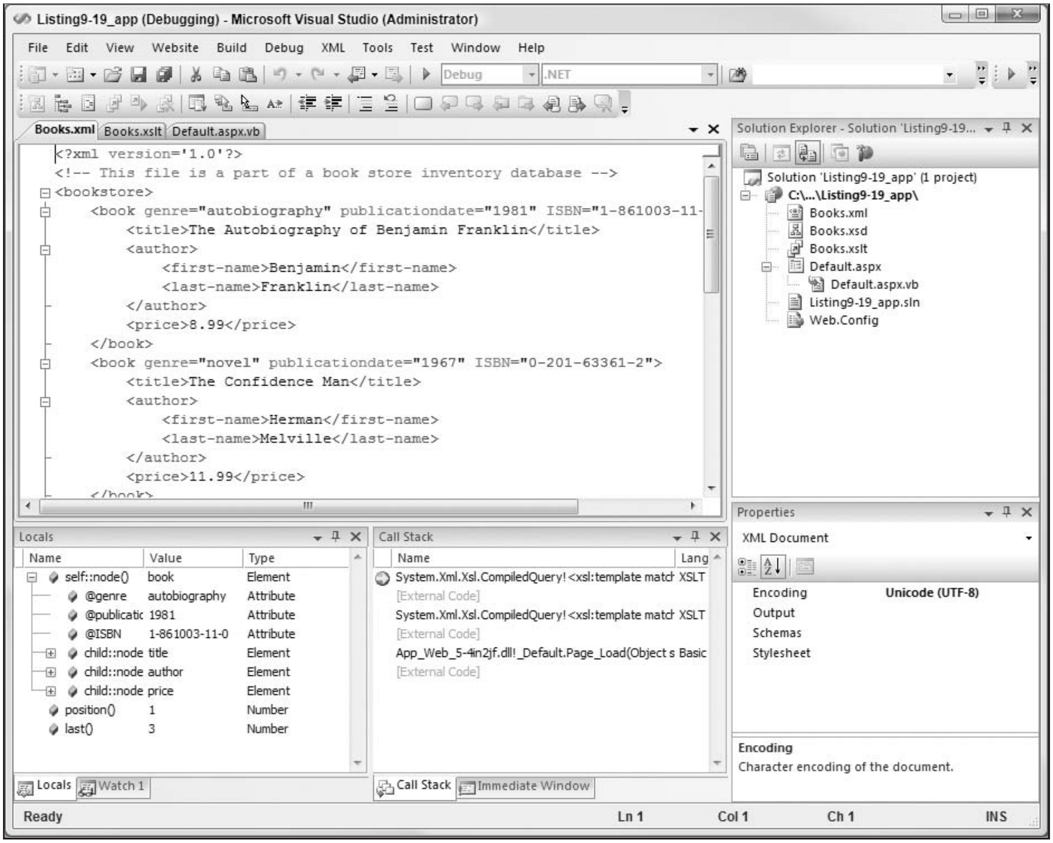
XSLT Debugging

One of the exciting new additions to ASP.NET 2.0 and Visual Studio was XSLT debugging. Visual Studio 2005 enabled breakpoints on XSLT documents and Visual Studio 2008 adds XSLT Debugger Data Breakpoints — the ability to break on nodes within the source XML document. However, be aware that XSLT debugging is available only in the Professional and Team System versions of Visual Studio and only when using the `XslCompiledTransform` class.

By passing the Boolean value `true` into the constructor of `XslCompiledTransform`, you can step into and debug your XSLT transformations within the Microsoft Development Environment.

```
Dim xslt As New XslCompiledTransform(True)
```

In Listing 10-19, change the call to the constructor of `XslCompiledTransform` to include the value `true` and set a breakpoint on the `Transform` method. When you reach that breakpoint, press F11 to step into the transformation. Figure 10-9 shows a debugging session of the `Books.xslt/Books.xml` transformation in process.


Listing9-19_app (Debugging) - Microsoft Visual Studio (Administrator)
File
Edit
View
Website
Build
Debug
XML
Tools
Test
Window
Help
Debug
.NET
Books.xml
Books.xslt
Default.aspx.vb
Solution Explorer - Solution 'Listing9-19...
Solution 'Listing9-19_app' (1 project)
C:\...\Listing9-19_app\
Books.xml
Books.xsd
Books.xslt
Default.aspx
Default.aspx.vb
Listing9-19_app.sln
Web.Config
Properties
Locals
Call Stack
Locals
Watch 1
Call Stack
Immediate Window
Ready
Ln 1
Col 1
Ch 1
INS

```
<?xml version='1.0'?>
<!-- This file is a part of a book store inventory database -->
<bookstore>
    <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-
        <title>The Autobiography of Benjamin Franklin</title>
        <author>
            <first-name>Benjamin</first-name>
            <last-name>Franklin</last-name>
        </author>
        <price>8.99</price>
    </book>
    <book genre="novel" publicationdate="1967" ISBN="0-201-63361-2">
        <title>The Confidence Man</title>
        <author>
            <first-name>Herman</first-name>
            <last-name>Melville</last-name>
        </author>
        <price>11.99</price>
    </book>
```

Name	Value	Type
self::node()	book	Element
@genre	autobiography	Attribute
@publicatic	1981	Attribute
@ISBN	1-861003-11-0	Attribute
child::node	title	Element
child::node	author	Element
child::node	price	Element
position()	1	Number
last()	3	Number

Name	Lang
System.Xml.Xsl.CompiledQuery!<xsl:template matcl	XSLT
[External Code]	
System.Xml.Xsl.CompiledQuery!<xsl:template matcl	XSLT
[External Code]	
App_Web_5-4in2jf.dll!_Default.Page_Load(Object s	Basic
[External Code]	

XML Document

Encoding	Unicode (UTF-8)
Output	
Schemas	
Stylesheet	

Encoding
Character encoding of the document.

Figure 10-9

In the past, debugging XSLT was largely an opaque process that required a third-party application to troubleshoot. The addition of debugging XSLT to Visual Studio means that your XML experience is just that much more integrated and seamless.

Databases and XML

You have seen that XML can come from any source whether it be a Web service, a file on disk, an XML fragment returned from a Web server, or a database. SQL server and ADO have rich support for XML, starting with the `ExecuteXmlReader` method of the `System.Data.SqlCommand` class. Additional support for XML on SQL Server 2000 is included with SQLXML 3.0 and its XML extensions, and SQL Server 2005 has native XML data type support built right in.

FOR XML AUTO

You can modify a SQL query to return XML with the `FOR XML AUTO` clause. If you take a simple query such as `select * from customers`, you just change the statement like so:

```
select * from customers FOR XML AUTO
```

`XML AUTO` returns XML fragments rather than a full XML document with a document element. Each row in the database becomes one element; each column in the database becomes one attribute on the element. Notice that each element in the following result set is named `Customers` because the `select` clause is `from customers`:

```
<Customers CustomerID="ALFKI" CompanyName="Alfreds Futterkiste"
ContactName="Maria Anders" ContactTitle="Sales Representative"
Address="Obere Str. 57" City="Berlin" PostalCode="12209"
Country="Germany" Phone="030-0074321" Fax="030-0076545" />

<Customers CustomerID="ANATR" CompanyName="Ana Trujillo Emparedados y
helados" ContactName="Ana Trujillo" ContactTitle="Owner" Address="Avda.
de la Constitucion 2222" City="Mexico D.F." PostalCode="05021"
Country="Mexico" Phone="(5) 555-4729" Fax="(5) 555-3745" />
```

If you add `ELEMENTS` to the query like so

```
select * from customers FOR XML AUTO, ELEMENTS
```

you get an XML fragment like this:

```
<Customers>
    <CustomerID>ALFKI</CustomerID>
    <CompanyName>Alfreds Futterkiste</CompanyName>
    <ContactName>Maria Anders</ContactName>
    <ContactTitle>Sales Representative</ContactTitle>
    <Address>Obere Str. 57</Address>
    <City>Berlin</City>
    <PostalCode>12209</PostalCode>
    <Country>Germany</Country>
    <Phone>030-0074321</Phone>
    <Fax>030-0076545</Fax>
</Customers>
<Customers>
    <CustomerID>ANATR</CustomerID>
    <CompanyName>Ana Trujillo Emparedados y helados</CompanyName>
    <ContactName>Ana Trujillo</ContactName>
    <ContactTitle>Owner</ContactTitle>
    <Address>Avda. de la Constitucion 2222</Address>
    <City>Mexico D.F.</City>
    <PostalCode>05021</PostalCode>
    <Country>Mexico</Country>
    <Phone>(5) 555-4729</Phone>
    <Fax>(5) 555-3745</Fax>
</Customers>
```

The previous example is just a fragment with no document element. To perform an XSLT transformation, you need a document element (sometimes incorrectly referred to as the ''root node''), and you probably want to change the `<Customers>` elements to `<Customer>`. By using an alias like `as Customer` in the `select` statement, you can affect the name of each row's element. The query `select * from Customers as Customer for XML AUTO, ELEMENTS` changes the name of the element to `<Customer>`.

Now, put together all the things you've learned from this chapter and create an `XmlDocument`, edit and manipulate it, retrieve data from SQL Server as an `XmlReader`, and style that information with XSLT into an HTML table all in just a few lines of code.

First, add a document element to the document retrieved by the SQL query `select * from Customers as Customer for XML AUTO, ELEMENTS`, as shown in Listing 10-21.

Listing 10-21: Retrieving XML from SQL Server 2000 using FOR XML AUTO

VB

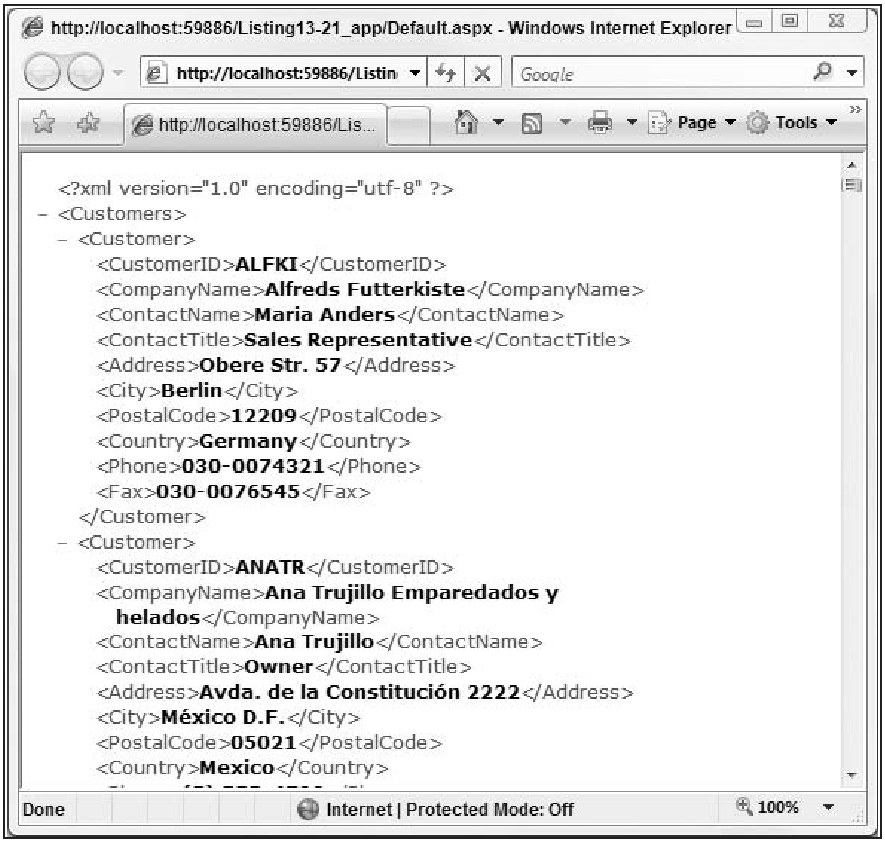
```
Dim connStr As String = "database=Northwind;Data Source=localhost;" & _
    " User id=sa;pwd=wrox"
Dim x As New XmlDocument()
Dim xpathnav As XPathNavigator = x.CreateNavigator()
Using conn As New SqlConnection(connStr)
    conn.Open()
    Dim command As New SqlCommand("select * from Customers as Customer " & _
        "for XML AUTO, ELEMENTS", conn)
    Using xw As XmlWriter = xpathnav.PrependChild()
        xw.WriteStartElement("Customers")
        Using xr As XmlReader = command.ExecuteXmlReader()
            xw.WriteNode(xr, True)
        End Using
        xw.WriteEndElement()
    End Using
End Using
Response.ContentType = "text/xml"
x.Save(Response.Output)
```

C#

```
string connStr = "database=Northwind;Data Source=localhost;User id=sa;pwd=wrox";
XmlDocument x = new XmlDocument();
XPathNavigator xpathnav = x.CreateNavigator();
using (SqlConnection conn = new SqlConnection(connStr))
{
    conn.Open();
    SqlCommand command = new SqlCommand(
        "select * from Customers as Customer for XML AUTO, ELEMENTS", conn);
    using (XmlWriter xw = xpathnav.PrependChild())
    {
        xw.WriteStartElement("Customers");
        using (XmlReader xr = command.ExecuteXmlReader())
        {
            xw.WriteNode(xr, true);
        }
```

```
            xw.WriteEndElement();
        }
    }
    Response.ContentType = "text/xml";
    x.Save(Response.Output);
```

This code creates an `XmlDocument` called `Customers`. Then it executes the SQL command and retrieves the XML data into an `XmlReader`. An `XPathNavigator` is created from the `XmlDocument`, and a child node is prepended to the document. A single call to the `WriteNode` method of the `XmlWriter` retrieved from the `XPathDocument` moves the entire XML fragment into the well-formed `XmlDocument`. Because the SQL statement contained `from Customers as Customer` as a table alias, each XML element is named `<Customer>`. Then, for this example, the resulting XML document is output directly to the `Response` object. You see the resulting XML in the browser shown in Figure 10-10.


http://localhost:59886/Listing13-21_app/Default.aspx - Windows Internet Explorer
http://localhost:59886/Listin
Google
http://localhost:59886/Lis...
Page
Tools
Done
Internet | Protected Mode: Off
100%

```
  <?xml version="1.0" encoding="utf-8" ?>
- <Customers>
  - <Customer>
      <CustomerID>ALFKI</CustomerID>
      <CompanyName>Alfreds Futterkiste</CompanyName>
      <ContactName>Maria Anders</ContactName>
      <ContactTitle>Sales Representative</ContactTitle>
      <Address>Obere Str. 57</Address>
      <City>Berlin</City>
      <PostalCode>12209</PostalCode>
      <Country>Germany</Country>
      <Phone>030-0074321</Phone>
      <Fax>030-0076545</Fax>
    </Customer>
  - <Customer>
      <CustomerID>ANATR</CustomerID>
      <CompanyName>Ana Trujillo Emparedados y
        helados</CompanyName>
      <ContactName>Ana Trujillo</ContactName>
      <ContactTitle>Owner</ContactTitle>
      <Address>Avda. de la Constitución 2222</Address>
      <City>México D.F.</City>
      <PostalCode>05021</PostalCode>
      <Country>Mexico</Country>
```

Figure 10-10

Of course, it's nice to see the resulting XML, but it's far more useful to style that information with XSLT. The XML Web Server control mentioned earlier is perfect for this task. However, in Listing 10-22, rather than setting both the `TransformSource` and `DocumentSource` properties as in Listing 10-25, you set only the `TransformSource` property at design time, and the `XmlDocument` is the one created in the code-behind of Listing 10-21.

Listing 10-22: The ASPX Page and XSLT to style the XML from SQL Server

ASPX

```
<%@ Page Language="C#" CodeFile="Default.aspx.cs" Inherits="Default_aspx" %>
<asp:xml id="Xml1" runat="server" transformsource="~/customersToHtml.xslt"/>
```

XSLT

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <h3>List of Customers</h3>
        <table border="1">
            <tr>
                <th>Company Name</th><th>Contact Name</th><th>Contact Title</th>
            </tr>
            <xsl:apply-templates select="//Customer"/>
        </table>
    </xsl:template>
    <xsl:template match="Customer">
        <tr>
            <td><xsl:value-of select="CompanyName"/></td>
            <td><xsl:value-of select="ContactName"/></td>
            <td><xsl:value-of select="ContactTitle"/></td>
        </tr>
    </xsl:template>
</xsl:stylesheet>
```

VB

```
'Response.ContentType = "text/xml"
'x.Save(Response.Output)
Xml1.XPathNavigator = xpathnav
```

C#

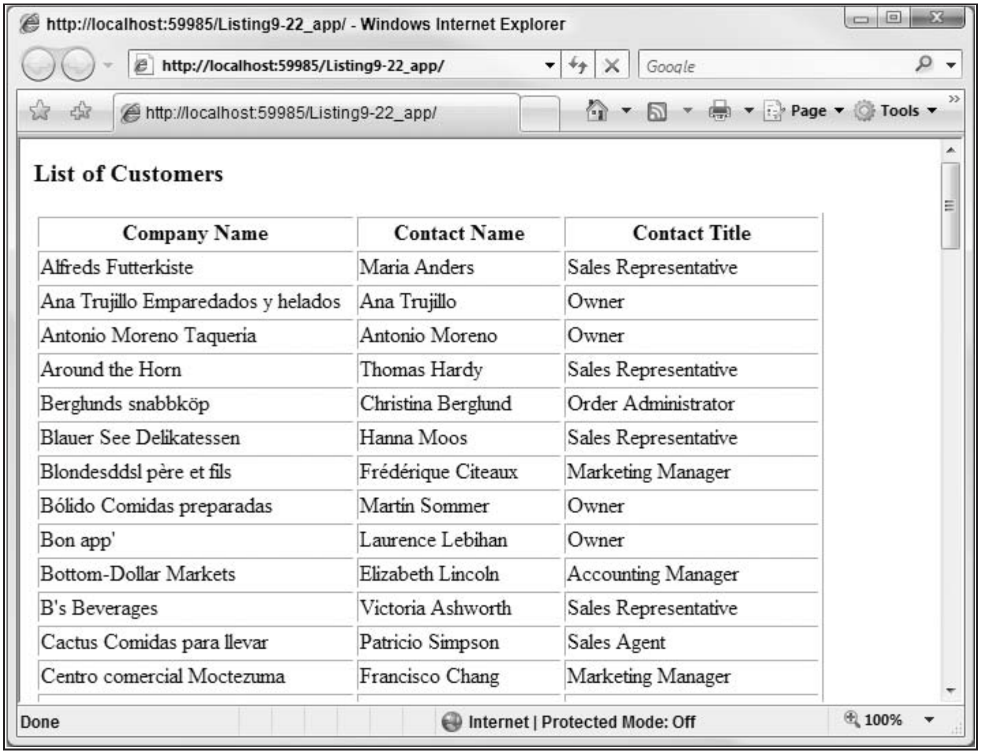
```
//Response.ContentType = "text/xml";
//x.Save(Response.Output);
Xml1.XPathNavigator = xpathnav;
```

In the code-behind file, the lines that set `ContentType` and write the XML to the `Response` object are commented out, and instead the `XPathNavigator` from the `XmlDocument` that is manipulated in Listing 10-21 is set as a property of the XML Web Server control. The control then performs the XSLT `Stylesheet` transformation, and the results are output to the browser, as shown in Figure 10-11.

You have an infinite amount of flexibility within the `System.Xml`, `System.Xml.Linq`, and `System.Data` namespaces. Microsoft has put together a fantastic series of APIs that interoperate beautifully. When you're creating your own APIs that expose or consume XML, compare them to the APIs that Microsoft has provided — if you expose your data over an `XmlReader` or `IXPathNavigable` interface, you are sure to make your users much happier. Passing XML around with these more flexible APIs (rather than as simple and opaque strings) provides a much more comfortable and intuitive expression of the XML information set.

Remember that the `XmlReader` that is returned from `SqlCommand.ExecuteXmlReader()` is holding its SQL connection open, so you must call `Close()`

when you're done using the `XmlReader`. The easiest way to ensure that this is done is the `using` statement. An `XmlReader` implements `IDisposable` and calls `Close()` for you as the variable leaves the scope of the `using` statement.


http://localhost:59985/Listing9-22_app/ - Windows Internet Explorer
http://localhost:59985/Listing9-22_app/
Google
http://localhost:59985/Listing9-22_app/
Page
Tools
Done
Internet | Protected Mode: Off
100%

List of Customers

Company Name	Contact Name	Contact Title
Alfreds Futterkiste	Maria Anders	Sales Representative
Ana Trujillo Emparedados y helados	Ana Trujillo	Owner
Antonio Moreno Taqueria	Antonio Moreno	Owner
Around the Horn	Thomas Hardy	Sales Representative
Berglunds snabbköp	Christina Berglund	Order Administrator
Blauer See Delikatessen	Hanna Moos	Sales Representative
Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager
Bólido Comidas preparadas	Martín Sommer	Owner
Bon app'	Laurence Lebihan	Owner
Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager
B's Beverages	Victoria Ashworth	Sales Representative
Cactus Comidas para llevar	Patricio Simpson	Sales Agent
Centro comercial Moctezuma	Francisco Chang	Marketing Manager

Figure 10-11

SQL Server 2005 and the XML Data Type

You've seen that retrieving data from SQL Server 2000 is straightforward, if a little limited. SQL Server 2005, originally codenamed Yukon, includes a number of very powerful XML-based features. Dare Obasanjo, a former XML Program Manager at Microsoft has said, ''The rise of the ROX [Relational-Object-XML] database has begun.'' SQL Server 2005 is definitely leading the way.

One of the things that is particularly tricky about mapping XML and the XML information set to the relational structure that SQL Server shares with most databases is that most XML data has a hierarchical structure. Relational databases structure hierarchical data with foreign key relationships. Relational data often has no order, but the order of the elements within `XmlDocument` is very important. SQL Server 2005 introduces a new data type called, appropriately, `XML`. Previously, data was stored in an `nvarchar` or other string-based data type. SQL Server 2005 can now have a table with a column of type `XML`, and each XML data type can have associated XML Schema.

The `FOR XML` syntax is improved to include the `TYPE` directive, so a query that includes `FOR XML TYPE` returns the results as a single XML-typed value. This XML data is returned with a new class called `System.Data.SqlXml`. It exposes its data as an `XmlReader` retrieved by calling `SqlXml.CreateReader`, so you'll find it to be very easy to use because it works like all the other examples you've seen in this chapter.

In a DataSet returned from SQL Server 2005, XML data defaults to being a string unless `DataAdapter.ReturnProviderSpecificTypes = true` is set or a schema is loaded ahead of time to specify the column type.

The `XML` data type stores data as a new internal binary format that is more efficient to query. The programmer doesn't have to worry about the details of how the XML is stored if it continues to be available on the XQuery or in `XmlReader`. You can mix column types in a way that was not possible in SQL Server 2000. You're used to returning data as either a DataSet or an `XmlReader`. With SQL Server 2005, you can return a DataSet where some columns contain XML and some contain traditional SQL Server data types.

Generating Custom XML from SQL 2005

You've seen a number of ways to programmatically generate custom XML from a database. Before reading this book, you've probably used `FOR XML AUTO` to generate fairly basic XML and then modified the XML in post processing to meet your needs. This was formerly a very common pattern. `FOR XML AUTO` was fantastically easy; and `FOR XML EXPLICIT`, a more explicit way to generate XML, was very nearly impossible to use.

SQL Server 2005 adds the new `PATH` method to `FOR XML`, which makes arbitrary XML creation available to mere mortals. SQL 2005's XML support features very intuitive syntax and very clean namespace handling.

Here is an example of a query that returns custom XML. The `WITH XMLNAMESPACES` commands at the start of the query set the stage by defining a default namespace and using column-style name aliasing to associate namespaces with namespace prefixes. In this example, `addr`: is the prefix for `urn:hanselman.com/northwind/address`.

```
use Northwind;
WITH XMLNAMESPACES (
 DEFAULT 'urn:hanselman.com/northwind'
 , 'urn:hanselman.com/northwind/address' as "addr"
)
SELECT CustomerID as "@ID",
           CompanyName,
           Address as "addr:Address/addr:Street",
           City as "addr:Address/addr:City",
           Region as "addr:Address/addr:Region",
           PostalCode as "addr:Address/addr:Zip",
           Country as "addr:Address/addr:Country",
           ContactName as "Contact/Name",
           ContactTitle as "Contact/Title",
           Phone as "Contact/Phone",
           Fax as "Contact/Fax"
FROM Customers
FOR XML PATH('Customer'), ROOT('Customers'), ELEMENTS XSINIL
```

The aliases using the `AS` keyword declaratively describe the elements and their nesting relationships, whereas the `PATH` keyword defines an element for the Customers table. The `ROOT` keyword defines the root node of the document.

The `ELEMENTS` keyword, along with `XSINIL`, describes how you handle null. Without these keywords, no XML element is created for a row's column that contains null; this absence of data in the database causes the omission of data in the resulting XML document. When the `ELMENTS XSINIL` combination is present, an element outputs using an explicit xsi:nil syntax such as `<addr:Region xsi:nil="true" />`.

When you run the example, SQL 2005 outputs an XML document like the one that follows. Note the namespaces and prefixes are just as you defined them.

```
<Customers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:addr="urn:hanselman.com/northwind/address"
xmlns="urn:hanselman.com/northwind">
  <Customer ID="ALFKI">
    <CompanyName>Alfreds Futterkiste</CompanyName>
    <addr:Address>
      <addr:Street>Obere Str. 57</addr:Street>
      <addr:City>Berlin</addr:City>
      <addr:Region xsi:nil="true" />
      <addr:Zip>12209</addr:Zip>
      <addr:Country>Germany</addr:Country>
    </addr:Address>
    <Contact>
      <Name>Maria Anders</Name>
      <Title>Sales Representative</Title>
      <Phone>030-0074321</Phone>
      <Fax>030-0076545</Fax>
    </Contact>
  </Customer>
...the rest of the document removed for brevity...
```

The resulting XML can now be manipulated using an `XmlReader` or any of the techniques discussed in this chapter.

Adding a Column of Untyped XML

SQL Server can produce XML from a query, and it can now also store XML in a single column. Because XML is a first-class data type within SQL Server 2005, adding a new column to the Customers table of the Northwind Database is straightforward. You can use any SQL Server management tool you like. I use the SQL Server Management Studio Express, a free download that can be used with any SQL SKU (including the free SQL Express 2005). Bring up your Query Analyzer or Management Studio Express and, with the Northwind database selected, execute the following query:

```
use Northwind;
BEGIN TRANSACTION
GO
ALTER TABLE dbo.Customers ADD
 Notes xml NULL
GO
COMMIT
```

Note the `xml` type keyword after `Notes` in the preceding example. If an XML Schema were already added to the database, you could add this new column and associate it with a named Schema Collection all at once using this syntax.

```
use Northwind;
BEGIN TRANSACTION
GO
ALTER TABLE dbo.Customers ADD
 Notes xml(DOCUMENT dbo.NorthwindCollection)
 GO
COMMIT
```

Here, the word `DOCUMENT` indicates that the column will contain a complete XML document. Use `CONTENT` to store fragments of XML that don't contain a root note. You haven't added a schema yet, so that's the next step. So far, you've added a `Notes` column to the Customers table that can be populated with prose. For example, a customer service representative could use it to describe interactions she's had with the customer, entering text into a theoretical management system.

Adding an XML Schema

Although the user could store untyped XML data in the Notes field, you should really include some constraints on what's allowed. XML data can be stored typed or untyped, as a fragment with no root note or as a document. Because you want store Customer interaction data entered and viewed from a Web site, ostensibly containing prose, XHTML is a good choice.

XML data is validated by XML Schemas, as discussed earlier in the chapter. However, SQL Server 2005 is a database, not a file system. It needs to store the schemas you want to reference in a location it can get to. You add a schema or schemas to SQL Server 2005 using queries formed like this:

```
CREATE XML SCHEMA COLLECTION YourCollection AS 'your complete xml schema here'
```

You'll be using the XHTML 1.0 Strict schema located on the W3C Web site shown here: `http://w3.org/TR/xhtml1-schema/#xhtml1-strict`. Copy the entire schema to a file, or download the schema directly from `http://w3.org/2002/08/xhtml/xhtml1-strict.xsd`.

When executing your query, you include the entire XSD inline in your schema. However, you should watch for few things. First, escape any single quotes so that `'` becomes `''` — that is, two single quotes, *not* one double — using a search and replace. Second, because SQL 2005 uses the MSXML6 XML parser to parse its XML, take into consideration a limitation in that parser. MSXML6 already has the `xml:` namespace prefix and associated namespace hard-coded internally, so you should remove the line from your schema that contains that namespace. This little oddity is documented, but buried within MSDN at `http://msdn2.microsoft.com/ms177489(en-US,SQL.90).aspx` and applies only to a few predefined schemas like this one that uses the `xml:` prefix and/or the `http://www.w3.org/XML/1998/namespace` namespace. In the fragment that follows, I've bolded the line you need to remove.

```
Use Northwind;
CREATE XML SCHEMA COLLECTION NorthwindCollection AS
'<?xml version="1.0" encoding="UTF-8"?>
<xs:schema version="1.0" xml:lang="en"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.w3.org/1999/xhtml"
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:xml="http://www.w3.org/XML/1998/namespace"
    elementFormDefault="qualified">
...the rest of the schema has been omitted for brevity...
</xs:schema>';
```

Instead, you want to execute a query like this, noting the single quote and semicolon at the very end.

```
Use Northwind;
CREATE XML SCHEMA COLLECTION NorthwindCollection AS
'<?xml version="1.0" encoding="UTF-8"?>
<xs:schema version="1.0" xml:lang="en"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.w3.org/1999/xhtml"
    xmlns="http://www.w3.org/1999/xhtml"
    elementFormDefault="qualified">
...the rest of the schema has been omitted for brevity...
</xs:schema>';
```

You may get a few schema validation warnings when you execute this query because of the complexity of the XHTML schema, but you can ignore them. Figure 10-12 shows the new NorthwindCollection schemas added to the Northwind database.

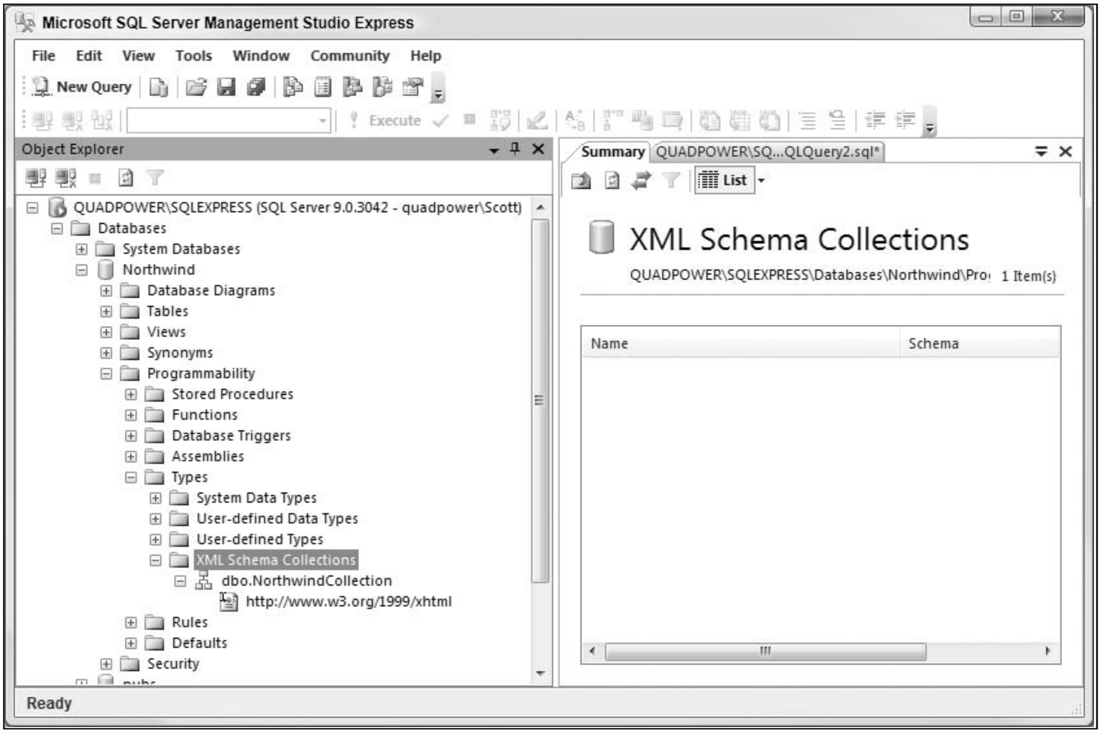

Microsoft SQL Server Management Studio Express
File
Edit
View
Tools
Window
Community
Help
New Query
Execute
Object Explorer
QUADPOWER\SQLEXPRESS (SQL Server 9.0.3042 - quadpower\Scott)
Databases
System Databases
Northwind
Database Diagrams
Tables
Views
Synonyms
Programmability
Stored Procedures
Functions
Database Triggers
Assemblies
Types
System Data Types
User-defined Data Types
User-defined Types
XML Schema Collections
dbo.NorthwindCollection
http://www.w3.org/1999/xhtml
Rules
Defaults
Security
Summary
QUADPOWER\SQ...QLQuery2.sql*
List
XML Schema Collections
QUADPOWER\SQLEXPRESS\Databases\Northwind\Pro
1 Item(s)
Name
Schema
Ready

Figure 10-12

Although Figure 10-12 shows the NorthwindCollection within the Object Explorer, you can also confirm that your schema has been added correctly using SQL, as shown in the example that follows:

```
Use Northwind;
SELECT XSN.name
FROM    sys.xml_schema_collections XSC
    JOIN sys.xml_schema_namespaces XSN ON
        (XSC.xml_collection_id = XSN.xml_collection_id)
```

The output of the query is something like the following. You can see that the XHTML namespace appears at the end along with the schemas that already existed in the system.

```
http://www.w3.org/2001/XMLSchema
http://schemas.microsoft.com/sqlserver/2004/sqltypes
http://www.w3.org/XML/1998/namespace
http://www.w3.org/1999/xhtml
```

Next you should associate the new column with the new schema collection. Using the Management Studio, you create one composite script that automates this process. In this case, however, you can continue to take it step by step so you see what's happening underneath.

Associating a XML Typed Column with a Schema

You can use the Microsoft SQLServer Management Studio Express to associate the NorthwindCollection with the new Notes column. Open the Customers table of the Northwind Database and, within its Column collection, right-click and select Modify. Select the Notes column, as shown in Figure 10-13. Within the Notes column's property page, open the XML Type Specification property and select the NorthwindCollection from the Schema Collection dropdown. Also, set the `Is XML Document` property to `Yes`.

At this point, you can save your table and a change script is generated and executed. If you want to see and save the change script after making a modification but before saving the changes, right-click in the grid and select Generate Change Script. Click the Save toolbar button or Ctrl+S to commit your changes to the Customers table.

Now that you've added a new Notes column and associated it with an XHTML schema, you're ready to add some data to an existing row.

Inserting XML Data into an XML Column

You start by adding some data to a row within the Northwind database's Customer table. Add some notes to the famous first row, the customer named Alfreds Futterkiste, specifically CustomerID ALFKI. You add or update data to a column with the XML type just as you add to any data type. For example, you can try an UPDATE query.

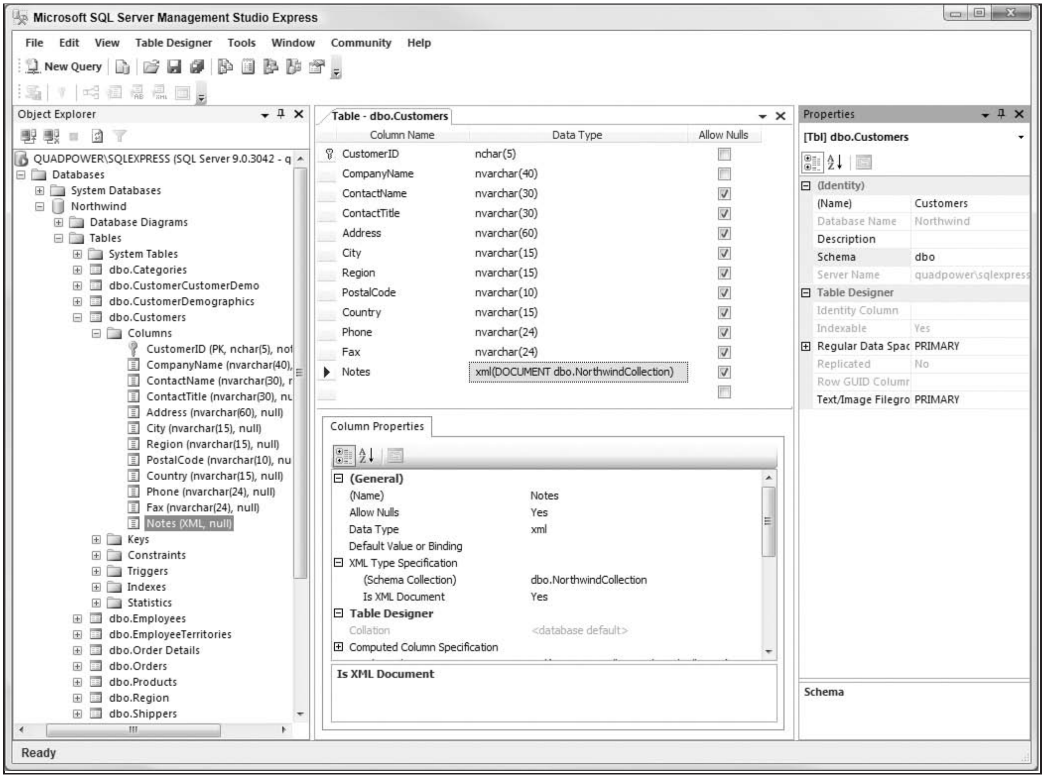
```
Use Northwind;
UPDATE Customers SET Notes = N'<HTML></HTML>' WHERE CustomerID = 'ALFKI';
```

Upon executing this query, you get this result:

```
Msg 6913, Level 16, State 1, Line 1
XML Validation: Declaration not found for element 'HTML'. Location: /*:HTML[1]
```

What's this? Oh, yes, you associated a schema with the XML column so the included document must conform, in this case, to the XHTML specification. Now, try again with a valid XHTML document that includes a correct namespace.

```
Use Northwind;
UPDATE Customers SET Notes = N'
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
 <head>
  <title>Notes about Alfreds</title>
 </head>
 <body>
  <p>He is a nice enough fellow.</p>
 </body>
</html>'
WHERE CustomerID = 'ALFKI';
```


Microsoft SQL Server Management Studio Express
File
Edit
View
Table Designer
Tools
Window
Community
Help
New Query
Object Explorer
QUADPOWER\SQLEXPRESS (SQL Server 9.0.3042 - q
Databases
System Databases
Northwind
Database Diagrams
Tables
System Tables
dbo.Categories
dbo.CustomerCustomerDemo
dbo.CustomerDemographics
dbo.Customers
Columns
CustomerID (PK, nchar(5), not
CompanyName (nvarchar(40),
ContactName (nvarchar(30), r
ContactTitle (nvarchar(30), nu
Address (nvarchar(60), null)
City (nvarchar(15), null)
Region (nvarchar(15), null)
PostalCode (nvarchar(10), nu
Country (nvarchar(15), null)
Phone (nvarchar(24), null)
Fax (nvarchar(24), null)
Notes (XML, null)
Keys
Constraints
Triggers
Indexes
Statistics
dbo.Employees
dbo.EmployeeTerritories
dbo.Order Details
dbo.Orders
dbo.Products
dbo.Region
dbo.Shippers
Table - dbo.Customers
Column Properties
Properties
Ready

Column Name	Data Type	Allow Nulls
CustomerID	nchar(5)	☐
CompanyName	nvarchar(40)	☐
ContactName	nvarchar(30)	☑
ContactTitle	nvarchar(30)	☑
Address	nvarchar(60)	☑
City	nvarchar(15)	☑
Region	nvarchar(15)	☑
PostalCode	nvarchar(10)	☑
Country	nvarchar(15)	☑
Phone	nvarchar(24)	☑
Fax	nvarchar(24)	☑
Notes	xml(DOCUMENT dbo.NorthwindCollection)	☑
		☐

(General)	
(Name)	Notes
Allow Nulls	Yes
Data Type	xml
Default Value or Binding	
XML Type Specification	
(Schema Collection)	dbo.NorthwindCollection
Is XML Document	Yes
Table Designer	
Collation	<database default>
Computed Column Specification	

Is XML Document

[Tbl] dbo.Customers

(Identity)	
(Name)	Customers
Database Name	Northwind
Description	
Schema	dbo
Server Name	quadpower\sqlexpress
Table Designer	
Identity Column	
Indexable	Yes
Regular Data Spac	PRIMARY
Replicated	No
Row GUID Columr	
Text/Image Filegro	PRIMARY

Schema

Figure 10-13

Execute this corrected query and you see a success message.

```
(1 row(s) affected)
```

After you've typed a column as XML and associated an XML Schema, SQL Server 2005 will allow only XML documents that validate. The data can be retrieved from SQL Server using standard System.Data techniques. It can be pulled out of a `DataReader` or a `DataSet` and manipulated with `XmlReader`s or as an `XmlDocument`.

Summary

XML and the XML InfoSet are both pervasive in the .NET Framework and in ASP.NET. All ASP.NET configuration files include associated XML Schema, and the Visual Studio Editor is even smarter about XML documents that use XSDs.

`XmlReader` and `XmlWriter` provide unique and incredibly fast ways to consume and create XML; they now also include even better support for mapping XML Schema types to CLR types, as well as other improvements. The `XmlDocument` and `XPathDocument` return in .NET 2.0 with API additions and numerous performance improvements, while the `XmlDataDocument` straddles the world of `System.Data` and `System.Xml`. ASP.NET and .NET 3.5 include support for XSLT via not only the new `XslCompiledTransform` but also command-line compilation, and tops it all with XSLT debugging support for compiled stylesheets. LINQ to XML introduces the new `System.Xml.Linq` namespace and supporting classes for a tightly integrated IntelliSense-supported coding experience. VB9 takes XML support to the next level with XML literals and XML namespace imports. The bridge classes and extension methods make the transition between `System.Xml` and `System.Xml.Linq` clean and intuitive.

All these ways to manipulate XML via the Base Class Library are married with XML support in SQL Server 2000 and 2005. SQL Server 2005 also includes the XML data type for storing XML in a first class column type validated by XML Schemas stored in the database.

11

IIS7

Internet Information Services 7.0 (IIS7) is the latest version of Microsoft’s Web Server. IIS7 has gone through significant architectural changes since the last version. The most notable change for ASP.NET developers is the deep integration of IIS7 and the ASP.NET Framework. This provides both ASP.NET developers and IIS7 administrators with an integrated programming environment that allows them to implement features and functionalities that were not possible before. This chapter will provide you with an overview of the IIS7 and ASP.NET integrated architecture and its constituent components, show you how to install, setup, and configure IIS7, as well as show you how to migrate your existing applications to IIS7.

Modular Architecture of IIS7

The main goal of the Microsoft IIS team for IIS 6.0 was to improve its security, performance, and reliability. For that reason, modularity and extensibility didn’t make it to the list of top priorities. That said, IIS 6.0 introduced a very important notion: selective disabling of features such as ISAPI extensions and standard CGI (Common Gateway Interface) components. One of the main problems with the earlier versions of IIS was that every feature had to be installed and enabled. There were no ways to disable specific features not needed by your application scenario.

IIS 6.0 enables only static file serving by default on a clean install of the Web server. In other words, dynamic features such as ISAPI extensions and CGI components are disabled by default unless the administrator explicitly enables them. Such customization of the Web server allows you to decrease the attack surface of your Web server, giving attackers fewer opportunities for attacks.

Disabling unwanted features was the first step toward making the IIS customizable. However, this step didn’t go far enough because IIS 6.0 still installs everything, which introduces the following problems:

- Disabled features consume server resources such as memory, and therefore increase the Web server footprint.
- Administrators still need to install service packs that address bugs in the disabled features, even though they’re never used.
- Administrators still need to install software updates for the disabled features.

In other words, administrators have to maintain the service features that are never used. All these problems stem from the fact that the architecture of IIS 6.0 is relatively monolithic. The main installation problem with a monolithic architecture is that it's based on an all-or-nothing paradigm where you have no choice but to install the whole system.

IIS 7.0 is modular to the core. Its architecture consists of over 40 feature modules from which you can choose. This allows you to install only feature modules you need to build a highly customized and very thin Web server. This provides the following important benefits:

- ❑ Decreases the footprint of your Web server.
- ❑ Administrators need to install only those service packs that address bugs in the installed feature modules.
- ❑ Administrators need to install software updates for only the installed feature modules.

In other words, administrators have to maintain and service only installed feature modules.

Next, will be an overview of the IIS7 feature modules or components that matter to the `ASP.NET` developer. These feature components are grouped into what is known as *functional areas*, where each functional area maps to a specific IIS package update. In other words, each package update contains one or more feature modules or components. Later you'll use these package updates to custom build your Web server.

> **You can find even more detailed technical information specific to IIS7 at `http://www.iis.net` or in Wrox's *Professional IIS7 and ASP.NET Integrated Programming* by Dr. Shahram Khosravi (2007) from which portions of this chapter are adapted.**

The top level IIS update is known as IIS-WebServerRole, and as the name suggests, the IIS-WebServerRole enables Windows Server 2008 and Windows Vista to adapt a Web server role, which enables them to exchange information over the Internet, an intranet, or an extranet. IIS-WebServerRole consists of these sub-roles:

- ❑ IIS-WebServer
- ❑ IIS-WebServerManagementTools
- ❑ IIS-FTPPulishingService

Roles depend on other roles and build a dependency hierarchy.

IIS-WebServer

The system will let you know when you're installing a new role whether that role will require new feature modules. For example, IIS-WebServer requires these modules:

- ❑ IIS-CommonHTTPFeatures
- ❑ IIS-ApplicationDevelopment
- ❑ IIS-HealthAndDiagnostics

- ❑ IIS-Security
- ❑ IIS-Performance

Let's take a brief look at the feature modules required by the main IIS-WebServer feature.

IIS-CommonHttpFeatures

The IIS-CommonHttpFeatures update contains the feature modules or components described in the following table:

Feature Module	**Description**
IIS-StaticContent	Use this module to enable your Web server to service requests for static content. Web site resources with file extensions such as `.html`, `.htm`, `.jpg`, and the like that can be serviced without server-side processing are known as static content.
IIS-DefaultDocument	This module allows you to specify a Web resource that will be used as the default resource when the request URL does not contain the name of the requested resource.
IIS-DirectoryBrowsing	Use this module to enable your Web server to display the contents of a specified directory to end users when they directly access the directory and no default document exists in the directory.
IIS-HttpErrors	Use this module to enable your Web server to support sending custom error messages to end users.
IIS-HttpRedirect	Use this module to enable your Web server to support request redirects.

IIS-ApplicationDevelopment

The IIS-ApplicationDevelopment update contains the feature modules that support different application types as described in the following table:

Feature Module	**Description**
IIS-ASPNET	Use this module to enable your Web server to host ASP.NET applications.
IIS-NetFxExtensibility	Use this module to enable your Web server to host managed modules.
IIS-ASP	Use this module to enable your Web server to host ASP applications.
IIS-CGI	Use this module to enable your Web server to support CGI executables.
IIS-ISAPIExtensions	Use this module to enable your Web server to use ISAPI extension modules to process requests.
IIS-ISAPIFilter	Use this module to enable your Web server to use ISAPI filter to customize the server behavior.
IIS-ServerSideIncludes	Use this module to enable your Web server to support `.stm`, `.shtm`, and `.shtml` include files.

IIS-HealthAndDiagnostics

The IIS-HealthAndDiagnostics package update contains the feature modules described in the following table:

Feature Module	Description
IIS-HttpLogging	Use this module to enable your Web server to log Web site activities.
IIS-LoggingLibraries	Use this module to install logging tools and scripts on your Web server.
IIS-RequestMonitor	Use this module to enable your Web server to monitor the health of the Web server and its sites and applications.
IIS-HttpTracing	Use this module to enable your Web server to support tracing for ASP.NET applications and failed requests.
IIS-CustomLogging	Use this module to enable your Web server to support custom logging for the Web server and its sites and applications.
IIS-ODBCLogging	Use this module to enable your Web server to support logging to an ODBC-compliant database.

IIS-Security

The IIS-Security package update contains the feature modules described in the following table:

Security Feature Module	Description
IIS-BasicAuthentication	Use this module to enable your Web server to support the HTTP 1.1 Basic Authentication scheme. This module authenticates user credentials against Windows accounts.
IIS-WindowsAuthentication	Use this module to enable your Web server to authenticate requests using NTLM or Kerberos.
IIS-DigestAuthentication	Use this module to enable your Web server to support the Digest Authentication scheme. The main difference between Digest and Basic is that Digest sends password hashes over the network as opposed to the passwords themselves.
IIS-ClientCertificateMapping-Authentication	Use this module to enable your Web server to authenticate client certificates with Active Directory accounts.
IIS-IISCertificateMapping-Authentication	Use this module to enable your Web server to map client certificates 1-to-1 or many-to-1 to a Windows security identity.
IIS-URLAuthorization	Use this module to enable your Web server to perform URL authorization
IIS-RequestFiltering	Use this module to enable your Web server to deny access based on specified configured rules.
IIS-IPSecurity	Use this module to enable your Web server to deny access based on domain name or IP address.

IIS-Performance

The following table describes the performance feature modules:

Performance Feature Module	Description
IIS-HttpCompressionStatic	Use this module to enable your Web server to compress static content before sending it to the client to improve the performance.
IIS-HttpCompressionDynamic	Use this module to enable your Web server to compress dynamic content before sending it to the client to improve the performance.

IIS-WebServerManagementTools

The following table describes the feature modules contained in the IIS-WebServerManagementTools update:

Feature Module	Description
IIS-ManagementConsole	This module installs the Web Server Management Console, which allows administration of local and remote IIS web servers.
IIS-Management-ScriptingTools	Use this module to enable your Web server to support local Web server management via IIS configuration scripts.
IIS-ManagementService	Use this module to enable your Web server to be managed remotely via Web Server Management Console.

The following table presents the feature modules in the IIS-IIS6ManagementCompatibility update:

Feature Module	Description
IIS-Metabase	Use this module to enable your Web server to support metabase calls to the new IIS7 configuration store.
IIS-WMICompatibility	Use this module to install the IIS 6.0 WMI scripting interfaces to enable your Web server to support these interfaces.
IIS-LegacyScripts	Use this module to install the IIS 6.0 configuration scripts, to enable your Web server to support these scripts.
IIS-LegacySnapIn	Use this module to install the IIS 6.0 Management Console to enable administration of remote IIS 6.0 servers from this computer.

IIS-FTPPublishingService

The feature modules contained in the IIS-FTPPublishingService package update are discussed in the following table:

> *At the time of this writing, Microsoft announced that they'd be releasing a significantly enhanced IIS7 FTP server for Windows Server 2008 and Vista as a separate download. You can get more information on this at* `http://go.microsoft.com/fwlink/?LinkId=75371`.

Feature Module	Description
IIS-FTPServer	Use this module to install the FTP service.
IIS-FTPManagement	Use this module to install the FTP Management Console.

Extensible Architecture of IIS7

IIS 6.0 allows you to extend the functionality of the Web server by implementing and plugging in your own custom ISAPI filter and extension modules. Unfortunately, ISAPI suffers from fundamental problems such as:

- ❑ Since ISAPI is not a convenient or friendly API, and writing an ISAPI filter or extension module is not an easy task to accomplish, it can take a lot of time and tends to be error-prone
- ❑ ISAPI is not a managed API, which means that ASP.NET developers cannot benefit from the rich features of the .NET Framework when they're writing ISAPI filter and extension modules

IIS 7.0 has replaced ISAPI with a new set of convenient object-oriented APIs that make writing new feature modules much easier. These APIs come in two different flavors: managed and native. The native API is a convenient C++ API that you can use to develop and plug native modules into the core Web server. The managed API, on the other hand, allows you to take full advantage of the .NET Framework and its rich environment. This allows both ASP.NET developers and IIS7 administrators to use convenient ASP.NET APIs to extend the core Web server.

IIS7 and ASP.NET Integrated Pipeline

Let's take a moment and talk about how IIS 6.0 and ASP.NET interact with each other. Both IIS 6.0 and ASP.NET have request processing pipelines. Each request processing pipeline is a pipeline of components that are invoked one after another to perform their specific request processing tasks. For example, both IIS 6.0 and ASP.NET pipelines contain an authentication component, which is called to authenticate the request, as shown in Figure 11-1.

A typical incoming HTTP request first goes through the IIS 6.0 pipeline. At some point along this pipeline, IIS 6.0 uses its metabase to map the request to a particular handler. The requests for ASP.NET resources such as ASP.NET pages are mapped to the `aspnet_isapi.dll` handler. This handler then loads the CLR and the target ASP.NET application, if they haven't already been loaded. This is where the ASP.NET request processing pipeline kicks in. To phrase it another way, the request ''jumps'' over into the ASP.NET world and continues through the ASP.NET pipeline.

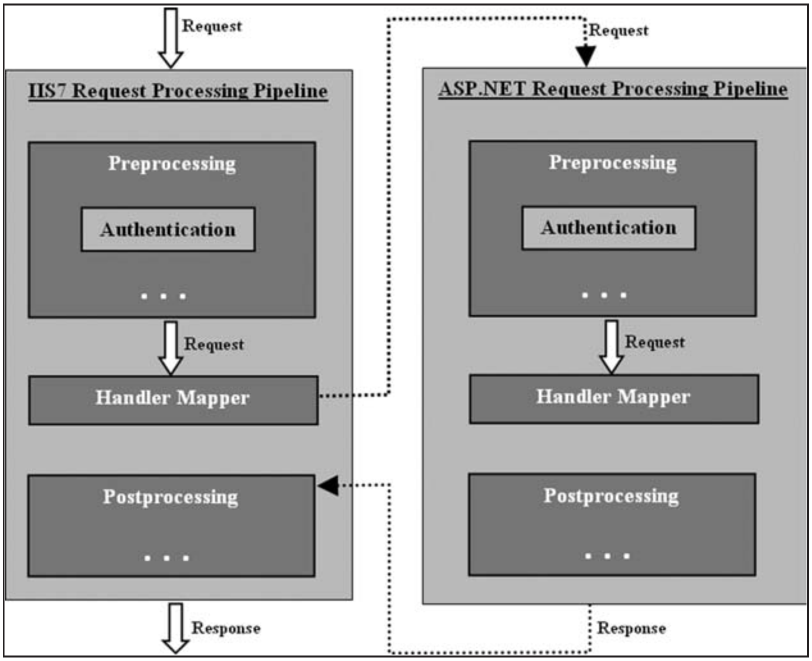

Request
IIS7 Request Processing Pipeline
Preprocessing
Authentication
. . .
Request
Handler Mapper
Postprocessing
. . .
Response
Request
ASP.NET Request Processing Pipeline
Preprocessing
Authentication
. . .
Request
Handler Mapper
Postprocessing
. . .
Response

Figure 11-1

At the beginning of the request, ASP.NET allows the components in its request processing pipeline to register one or more event handlers for one or more ASP.NET application-level events. ASP.NET then fires these events one after another and calls these event handlers to allow each component to perform its specific request processing task. At some point along the pipeline, ASP.NET uses the configuration file to map the request to a particular handler. The main responsibility of the handler is to process the request and generate the appropriate markup text, which will then be sent back to the requesting browser.

Having *two separate pipelines*, that is, IIS 6.0 and ASP.NET pipelines, working on the same request introduces the following problems:

- There's a fair amount of duplication. For example, both pipelines contain an authentication component, which means that the same request gets authenticated twice.
- Because the ASP.NET pipeline begins after the IIS pipeline maps the request to the `aspnet_isapi` extension module, the ASP.NET pipeline has no impact on the IIS pipeline steps prior to handler mapping.
- Because the rest of the IIS pipeline steps don't occur until the ASP.NET pipeline finishes, the ASP.NET pipeline has no impact on these IIS pipeline steps either.
- Because the ASP.NET pipeline comes into play when the IIS pipeline maps the request to the `aspnet_isapi` extension module, and because this mapping is done only for requests to ASP.NET content, the ASP.NET pipeline components cannot be applied to requests to non-ASP.NET content such as `.jpg`, `.js`, `.asp`, CGI, and the like. For example, you cannot easily use the ASP.NET authentication and authorization modules to protect the non-ASP.NET contents of your application without a significant performance penalty under IIS6.

IIS7 has changed all that by removing the `aspnet_isapi` extension module and combining the ASP.NET 3.5 and IIS pipelines into a **single integrated request processing pipeline**.

This new integrated design resolves all the previously mentioned problems as follows:

- The integrated pipeline does not contain any duplicate components. For example, the request is authenticated once.
- The ASP.NET modules are now first-class citizens in the integrated pipeline. They can come before, replace, or come after any native IIS7 modules. This allows ASP.NET to intervene at any stage of the request processing pipeline.
- Because the integrated pipeline treats managed modules like native modules, you can apply your ASP.NET managed modules to non-ASP.NET content. For example, you can use the ASP.NET authentication and authorization modules to protect the non-ASP.NET contents of your application, such as `asp` pages much easier than IIS6 and without the performance penalities.

Note however that when IIS7 is processing requests for ASP.NET content there are two different potential request processing pipelines: IIS7 ''Integrated'' and ASP.NET ''Classic''. The Classic pipeline basically puts IIS7 into ''IIS 6.0'' pipeline mode for a particular Application Pool. We'll see more on that when we configure an application pool later in this chapter.

Building a Customized Web Server

To understand IIS7, let's start by setting it up on a fresh system. You can use Windows Vista or Windows Server 2008 for this exercise.

Remember that IIS7 setup is completely modular, allowing you to custom build your Web server from a list of over 40 available feature modules. This ensures that your Web server contains only the feature modules you need, thereby decreasing the attack surface and footprint of your server. In this section, you'll walk through the steps that you need to take to build your very own custom Web server on Windows Vista (including Windows Vista Home Premium, Windows Vista Professional, and Windows Vista Ultimate editions) and Windows Server 2008 operating systems.

In general, there are five different IIS7 setup options:

- Windows Features dialog (Windows Vista only)
- Server Manager tool (Windows Server 2008 only)
- `pkgmgr.exe` command line tool (both Windows Vista and Windows Server 2008)
- Unattended (both Windows Vista and Windows Server 2008)
- Upgrade (both Windows Vista and Windows Server 2008)

Before drilling down into the details of these five setup options, you need to understand the dependencies between the installable updates.

Update Dependencies

When you're installing an update, you must also install the updates that it depends on. In general, there are two types of dependencies: interdependencies and parent-dependencies. The following table presents the update interdependencies:

Update	**Depends On**
IIS-WebServer	WAS-ProcessModel
IIS-ASP	IIS-ISAPIExtensions IIS-RequestFiltering
IIS-ASPNET	IIS-DefaultDocument IIS-NetFxExtensibility WAS-NetFxEnvironment IIS-ISAPIExtensions IIS-ISAPIFilter IIS-RequestFiltering
IIS-NetFxExtensibility	WAS-NetFxEnvironment IIS-RequestFiltering
IIS-ManagementService	IIS-WebServer IIS-ManagementConsole WAS-NetFxEnvironment WAS-ConfigurationAPI
IIS-ManagementConsole	WAS-ConfigurationAPI
IIS-ManagementScriptingTools	WAS-ConfigurationAPI
IIS-LegacyScripts	IIS-Metabase IIS-WMICompatibility

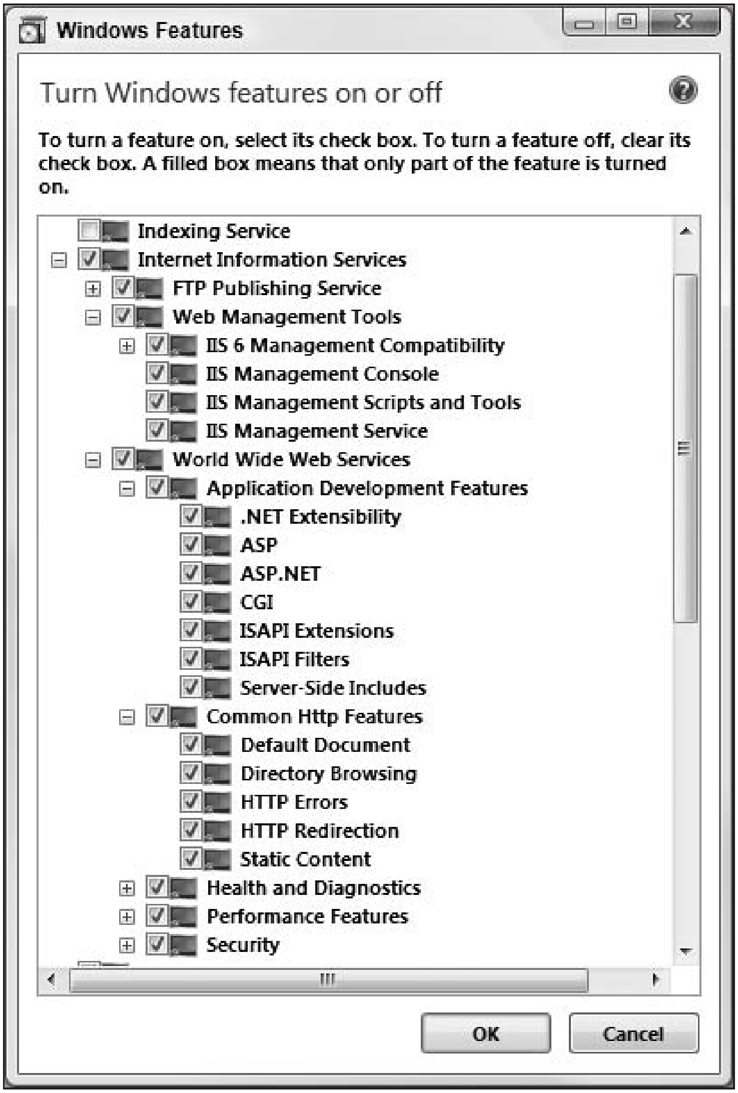
Every update also depends on its parent update. For example, to install IIS-WebServer, you must also install its parent update, IIS-WebServerRole.

Installing IIS7 on Windows Vista

Under Windows Vista, you install IIS7 from the Programs and Features application and click Turn Windows Features on or off. This dialog does an excellent job illustrating the hierarchy of modules available within IIS7, as shown in Figure 11-2.

Installing IIS7 on Windows Server 2008

You install IIS7 on Window Server 2008 by adding the IIS Server Role from the Server Manager as shown in Figure 11-3. In a clean install of the Windows Server 2008, the server is originally in no roles. The role that you're interested in is the Web Server role. Recall that this is the role that allows the server to share information on the Internet, an intranet, or an extranet. The first order of business is to launch the Add Roles Wizard from the Server Manager to add this role to your server.



Windows Features

Turn Windows features on or off

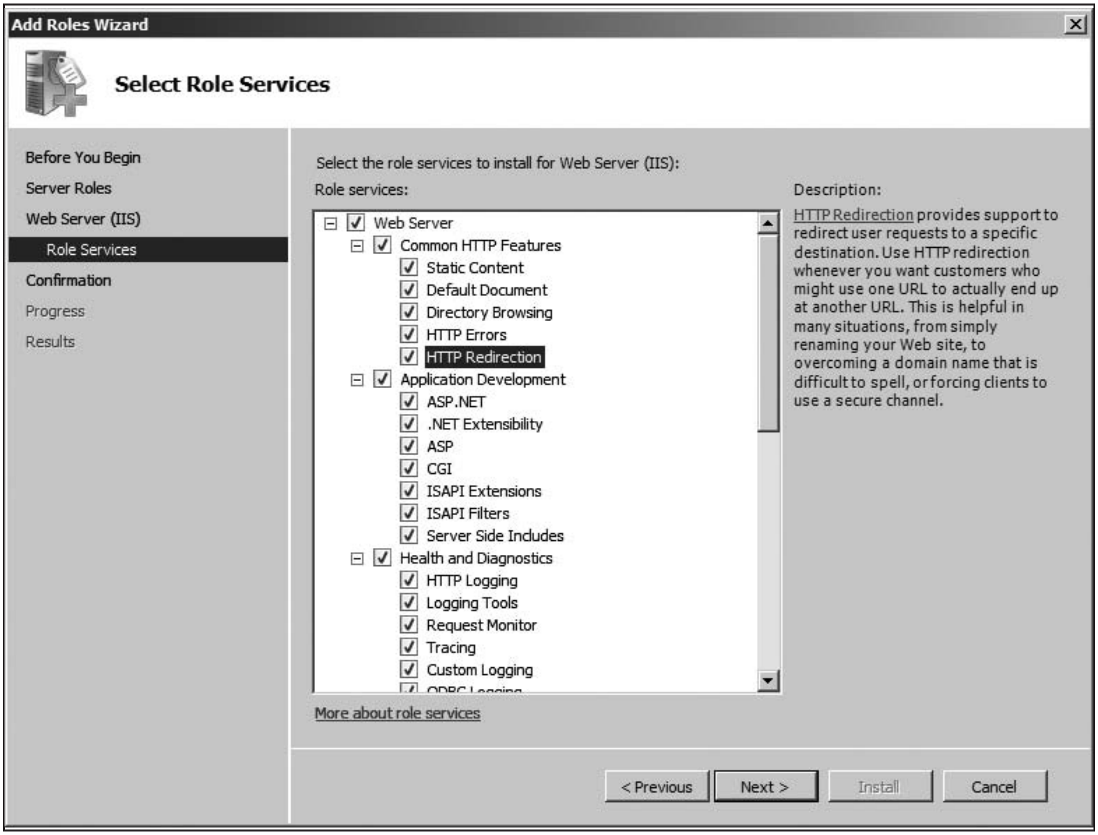
To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- ☐ Indexing Service
- ☑ Internet Information Services
  - ☑ FTP Publishing Service
  - ☑ Web Management Tools
    - ☑ IIS 6 Management Compatibility
    - ☑ IIS Management Console
    - ☑ IIS Management Scripts and Tools
    - ☑ IIS Management Service
  - ☑ World Wide Web Services
    - ☑ Application Development Features
      - ☑ .NET Extensibility
      - ☑ ASP
      - ☑ ASP.NET
      - ☑ CGI
      - ☑ ISAPI Extensions
      - ☑ ISAPI Filters
      - ☑ Server-Side Includes
    - ☑ Common Http Features
      - ☑ Default Document
      - ☑ Directory Browsing
      - ☑ HTTP Errors
      - ☑ HTTP Redirection
      - ☑ Static Content
    - ☑ Health and Diagnostics
    - ☑ Performance Features
    - ☑ Security

Figure 11-2

Notice that the same familiar check box list feature hierarchy exists in both Windows Vista and Windows Server 2008.

As you make selections the system will prompt you for dependent features as they are needed. For example, if you select ASP.NET you are prompted to add .Net Extensibility. For a Windows Server 2008 pure development machine, we recommend that you add Application Development, Health and Diagnostics and Security.


Add Roles Wizard
Select Role Services
Before You Begin
Server Roles
Web Server (IIS)
Role Services
Confirmation
Progress
Results
Role services:
Description:
More about role services
< Previous
Next >
Install
Cancel

Select the role services to install for Web Server (IIS):

- ☑ Web Server
  - ☑ Common HTTP Features
    - ☑ Static Content
    - ☑ Default Document
    - ☑ Directory Browsing
    - ☑ HTTP Errors
    - ☑ HTTP Redirection
  - ☑ Application Development
    - ☑ ASP.NET
    - ☑ .NET Extensibility
    - ☑ ASP
    - ☑ CGI
    - ☑ ISAPI Extensions
    - ☑ ISAPI Filters
    - ☑ Server Side Includes
  - ☑ Health and Diagnostics
    - ☑ HTTP Logging
    - ☑ Logging Tools
    - ☑ Request Monitor
    - ☑ Tracing
    - ☑ Custom Logging

HTTP Redirection provides support to redirect user requests to a specific destination. Use HTTP redirection whenever you want customers who might use one URL to actually end up at another URL. This is helpful in many situations, from simply renaming your Web site, to overcoming a domain name that is difficult to spell, or forcing clients to use a secure channel.

Figure 11-3

Command-Line Setup Options

Windows Vista and Windows Server 2008 come with a new command line tool named `pkgmgr.exe` that you can use to custom install IIS7. The following table describes the available options on this command-line tool:

Option	Description
`/iu:update1;update2...`	Run the tool with this option to install the specified updates. Notice that the update list contains a semi-colon separated list of the update names discussed in the previous sections.
`/uu:update1;update2...`	Run the tool with this option to uninstall the specified updates. Notice that the update list contains a semi-colon separated list of update names discussed in the previous sections.
`/n:unattend.xml`	Run the tool with this option to install or uninstall the updates specified in the specified `unattend.xml` file. You'll learn about this file in the following section.

When you use the `pkgmgr.exe` command-line tool to install specified updates, you must also explicitly specify and install the updates that your specified updates depend on. For example, if you decide to install the IIS-CommonHttpFeatures update, you must also install its parent update, that is, IIS-WebServer. To install the IIS-WebServer update you must also install its parent update, IIS-WebServerRole, and the update that it depends on, WAS-ProcessModel (see the Update Dependencies table). To install the WAS-ProcessModel update you must also install its parent update, WAS-WindowsActivationService update:

```
start /w pkgmgr.exe /iu:IIS-WebServerRole;WAS-WindowsActivationService;
WAS-ProcessModel;
IIS-WebServer;IIS-CommonHttpFeatures
```

Notice that if you don't specify the `start /w` option, the command-line tool will return immediately and process everything in the background, which means that you won't be able to see when the setup is completed.

Unattended Setup Option

As mentioned earlier, the `pkgmgr.exe` command line tool comes with the `/n:unattend.xml` option. `unattend.xml` is the XML file that contains the updates to be installed or uninstalled. This XML file provides you with two benefits. First, you don't have to directly enter the names of the updates on the command line. Second, you can store this file somewhere for reuse in other Web server machines. This XML file must have the same schema as the XML file shown in Listing 11-1. This listing installs the IIS-CommandHttpFeatures update and the updates that it depends on as discussed in the previous section.

Listing 11-1: The unattend.xml file

```
<?xml version="1.0" ?>
<unattend xmlns="urn:schemas-microsoft-com:unattend"
xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State">
  <servicing>
    <!-- Install a selectable update in a package that is in the
     Windows Foundation namespace -->
    <package action="configure">
      <assemblyIdentity name="Microsoft-Windows-Foundation-Package"
       version="6.0.5308.6" language="neutral" processorArchitecture="x86"
       publicKeyToken="31bf3856ad364e35" versionScope="nonSxS" />

      <selection name="IIS-WebServerRole" state="true"/>
      <selection name="WAS-WindowsActivationService" state="true"/>
      <selection name="WAS-ProcessModel" state="true"/>
      <selection name="IIS-WebServer" state="true"/>
      <selection name="IIS-CommonHttpFeatures" state="true"/>
    </package>
  </servicing>
</unattend>
```

Notice that the `< servicing >` element contains one or more `<package>` elements that contain `<selection>` child elements, and each child element specifies a particular update. The `<selection>` child element features two attributes named `name` and `state`. The `name` attribute contains the update name to be installed or uninstalled. Set the `state` attribute to `true` to install or `false` to uninstall the specified update.

Upgrade

If you're upgrading from the Windows XP to Windows Vista, or from Windows Server 2003 to Windows Server 2008, and if your old operating system has IIS installed, Windows Vista or Windows Server 2008 setup automatically scans through the capabilities of the installed IIS and ensures that the new install of IIS7 supports those features and capabilities. Unfortunately, due to the monolithic architecture of IIS 5.1 and IIS 6.0, this installation ends up installing almost all of the feature modules of IIS7. I highly recommend that after the upgrade you use one of the previously discussed installation options to uninstall the updates that you do not need to decrease the attack surface and footprint of your Web server.

Internet Information Services (IIS) Manager

In this section I'll walk you through different features of the IIS Manager. There are two ways to launch the IIS Manager: GUI-based and command line. If you feel more comfortable with a GUI-based approach, follow these steps to launch the IIS7 Manager:

1. Launch the Control Panel
2. Click System and Maintenance
3. Click Administrative Tools
4. Click the Internet Information Services (IIS) Manager

If you feel more comfortable with command line tools, use the following command line to launch the IIS Manager:

```
%windir%\system32\inetsrv\inetmgr.exe
```

You can also just type **IIS** into the new Start menu. Make sure to run the IIS7 Manager and not the legacy IIS6 Manager. Note you'll need administration privileges to launch the IIS Manager. If you don't login with the built-in Administrator account, when you try to launch the IIS Manager, Windows launches a dialog. The content of this dialog depends on whether your account has administration privileges. If it does, the dialog simply asks you to confirm the requested action. If it doesn't, the dialog asks for the administrative credentials.

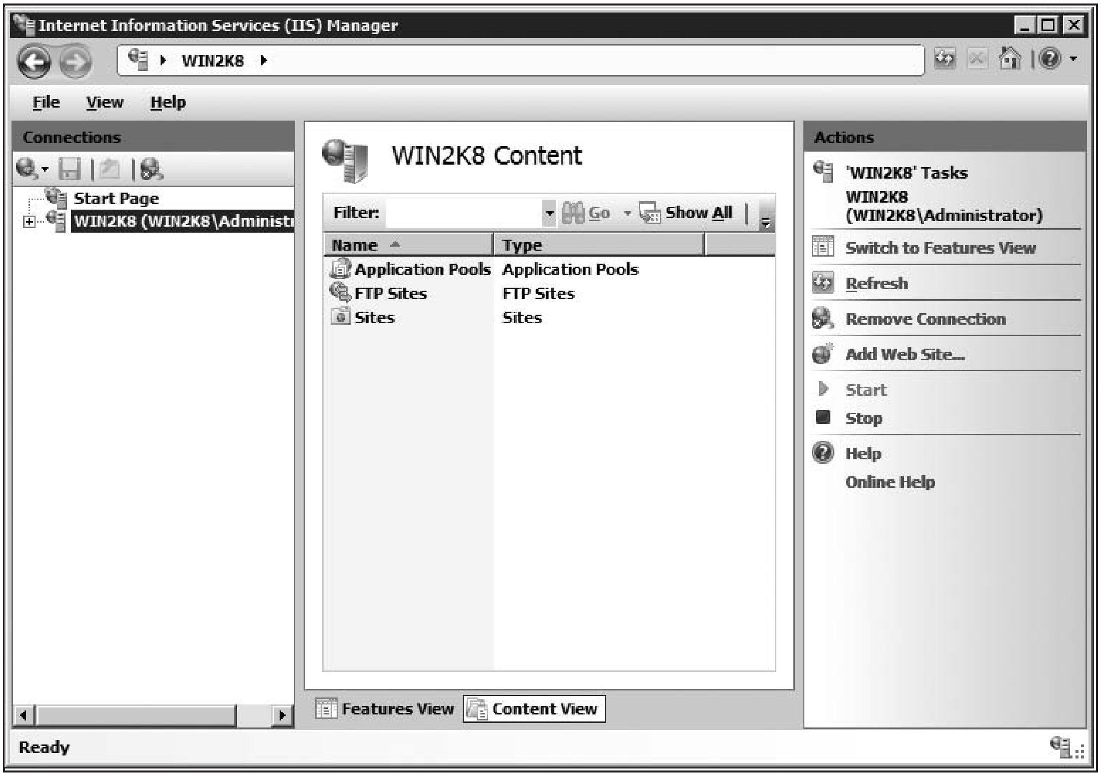
As Figure 11-4 shows, the IIS Manager consists of three panes. The first pane, which is known as the Connections pane, contains a node that represents the Web server. This node has two child nodes:

- ❑ Application Pools
- ❑ Sites. The label of this node is ''Sites'' on Windows Server 2008 and ''Web Sites'' on Windows Vista.

The second pane, which is known as workplace pane, consists of these two tabs:

- ❑ **Features View:** If you select a node in the Connections pane, the Features View tab will allow you to edit the features associated with the selected node.
- ❑ **Content View:** If you select a node in the Connections pane, the Content View tab will display all the child nodes of the selected node.

The third pane, which is known as Actions pane, contains a bunch of links where each link performs a particular task on the node selected in the first or second pane.


Internet Information Services (IIS) Manager
WIN2K8
File
View
Help
Connections
Start Page
WIN2K8 (WIN2K8\Administr
WIN2K8 Content
Filter:
Go
Show All
Features View
Content View
Actions
'WIN2K8' Tasks
WIN2K8 (WIN2K8\Administrator)
Switch to Features View
Refresh
Remove Connection
Add Web Site...
Start
Stop
Help
Online Help
Ready

Name	Type
Application Pools	Application Pools
FTP Sites	FTP Sites
Sites	Sites

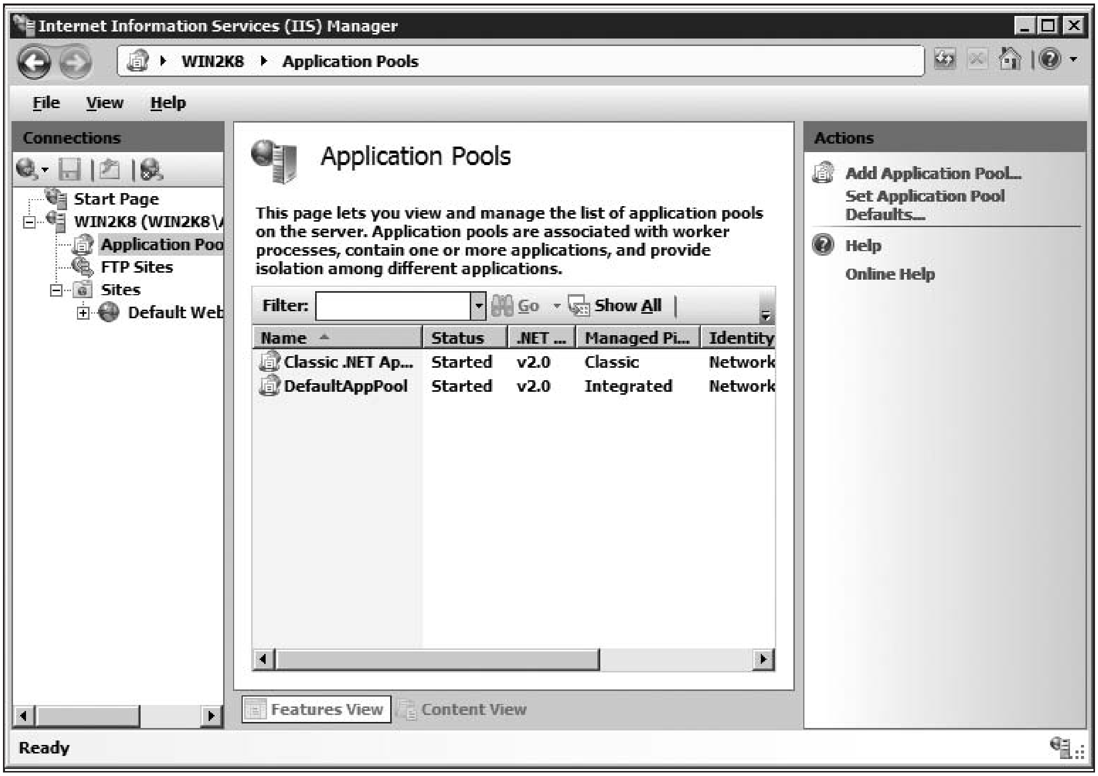
Figure 11-4

Application Pools

Now click the Application Pools node in the Connections pane to display the available application pools as shown in Figure 11-5.

Notice that the Actions pane contains an Add Application Pool link. Click this link to launch the dialog shown in Figure 11-6. This dialog allows you to add a new application pool and to specify its name. It also allows you to specify the .NET version that will be loaded into the application pool. Remember, all ASP.NET applications in the same application pool must use the same .NET version because. NET runtimes of differing versions cannot be loaded into the same worker process.

The Managed pipeline mode drop-down list on this dialog contains two options, Integrated and Classic, as shown in Figure 11-6. This specifies whether the IIS should run in Integrated or Classic mode for this application pool. All applications in the same application pool use the same IIS mode.

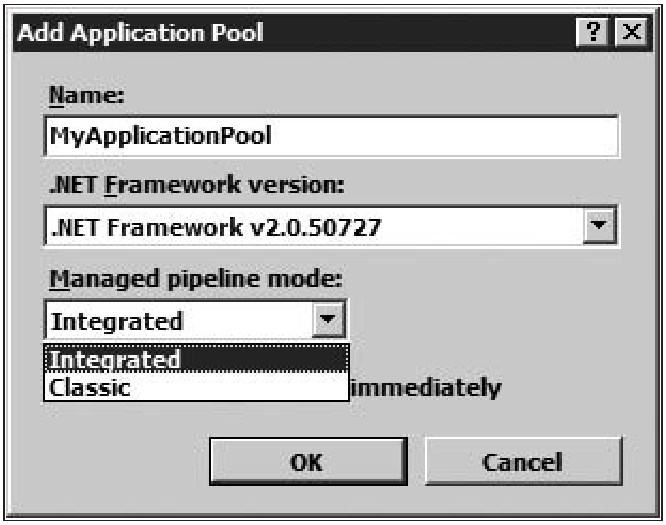

Internet Information Services (IIS) Manager
WIN2K8
Application Pools
File
View
Help
Connections
Start Page
WIN2K8 (WIN2K8\
Application Poo
FTP Sites
Sites
Default Web
Actions
Add Application Pool...
Set Application Pool Defaults...
Help
Online Help
Filter:
Go
Show All
Features View
Content View
Ready

Application Pools

This page lets you view and manage the list of application pools on the server. Application pools are associated with worker processes, contain one or more applications, and provide isolation among different applications.

Name	Status	.NET ...	Managed Pi...	Identity
Classic .NET Ap...	Started	v2.0	Classic	Network
DefaultAppPool	Started	v2.0	Integrated	Network

Figure 11-5


Add Application Pool
Integrated
Classic
immediately
OK
Cancel

Name: MyApplicationPool

.NET Framework version: .NET Framework v2.0.50727

Managed pipeline mode: Integrated

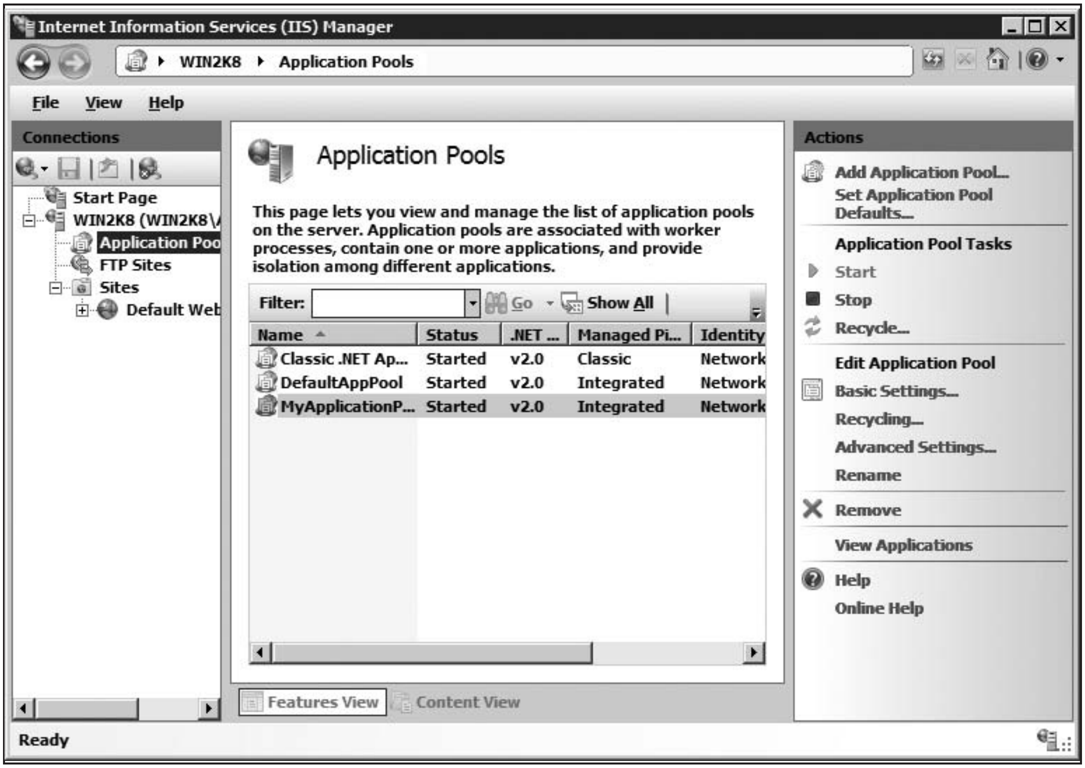
Figure 11-6

After making your selection click OK to commit the changes. Now open the `applicationHost.config` file in `%windir%\system32\inetsrv\config`. You'll need to be an administrator in order to see this file, and you might find it easiest to look for it from an Administrative Command Prompt. You should see the highlighted section shown in Listing 11-2.

Listing 11-2: The applicationHost.config file

```
<system.applicationHost>
  <applicationPools>
    . . .
     <add name="MyApplicationPool" />
     . . .
   </applicationPools>
 </system.applicationHost>
```

Click the newly created MyApplicationPool node in the middle pane. You should see new links on the Actions pane, which allow you to edit the properties of the application pool as shown in Figure 11-7.


Internet Information Services (IIS) Manager
WIN2K8
Application Pools
File
View
Help
Connections
Start Page
WIN2K8 (WIN2K8\A
Application Poo
FTP Sites
Sites
Default Web
Application Pools
Filter:
Go
Show All
Features View
Content View
Ready
Actions
Add Application Pool...
Set Application Pool Defaults...
Application Pool Tasks
Start
Stop
Recycle...
Edit Application Pool
Basic Settings...
Recycling...
Advanced Settings...
Rename
Remove
View Applications
Help
Online Help

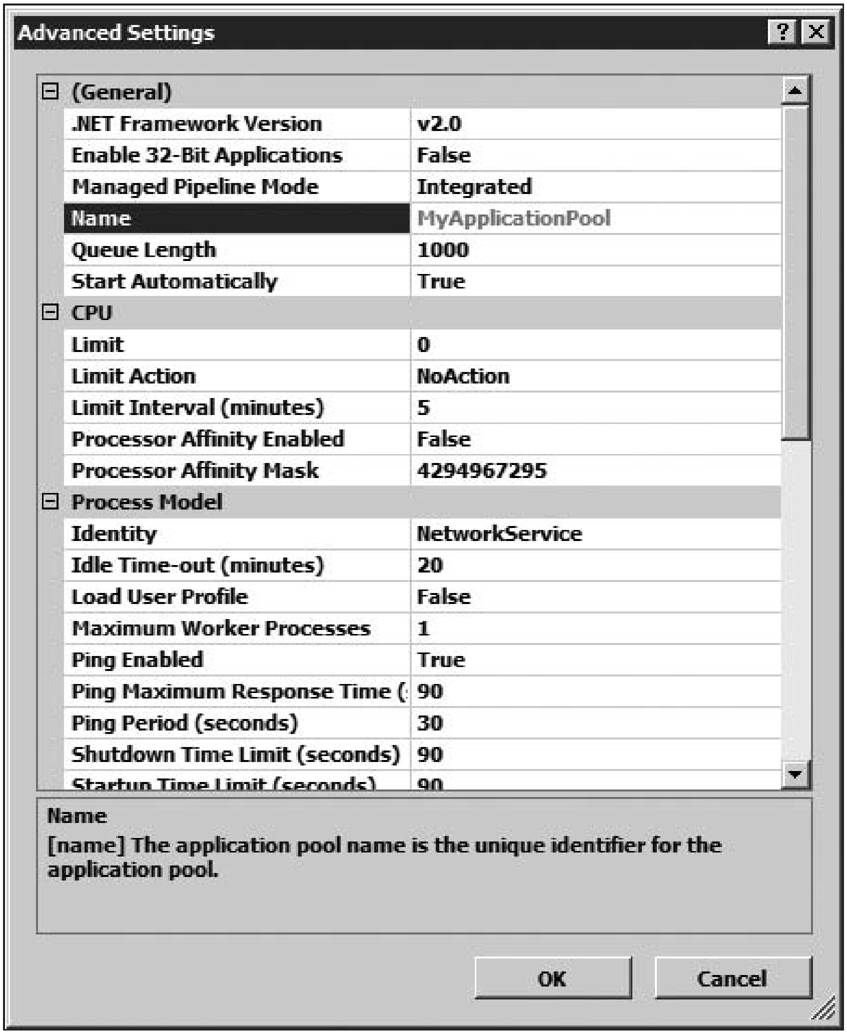
This page lets you view and manage the list of application pools on the server. Application pools are associated with worker processes, contain one or more applications, and provide isolation among different applications.

Name	Status	.NET ...	Managed Pi...	Identity
Classic .NET Ap...	Started	v2.0	Classic	Network
DefaultAppPool	Started	v2.0	Integrated	Network
MyApplicationP...	Started	v2.0	Integrated	Network

Figure 11-7

Click the Advanced Settings link to launch the Advanced Settings dialog shown in Figure 11-8. Notice that all settings of the newly created application pool have default values. However, as Listing 11-2 shows, none of these values show up in the `applicationHost.config` file. Where are these values stored? As you'll see later, the new IIS7 configuration system maintains the schema of the `applicationHost.config` file in two files named `ASPNET_schema.xml` and `IIS_schema.xml`. These schema files also specify and store the default values for configuration sections, including the `<applicationPools>` section. Storing the default configuration settings in one location as opposed to adding them to every single `<add>` element that represents an application pool keeps the configuration files small and more readable.

If you're running a 64-bit OS, make note of the Enable 32-bit Applications option in Advanced Settings. By default, ASP.NET applications will run as 64-bit on 64-bit OSes unless you switch them to 32-bit explicitly. Ninety-nine percent of managed applications will run fine as 64-bit, but if your application is calling into unmanaged code like COM objects or DLLs via P/Invoke, you might need to explicitly set aside an application pool for your 32-bit application.



⊟ (General)	
.NET Framework Version	v2.0
Enable 32-Bit Applications	False
Managed Pipeline Mode	Integrated
Name	MyApplicationPool
Queue Length	1000
Start Automatically	True
⊟ CPU	
Limit	0
Limit Action	NoAction
Limit Interval (minutes)	5
Processor Affinity Enabled	False
Processor Affinity Mask	4294967295
⊟ Process Model	
Identity	NetworkService
Idle Time-out (minutes)	20
Load User Profile	False
Maximum Worker Processes	1
Ping Enabled	True
Ping Maximum Response Time (	90
Ping Period (seconds)	30
Shutdown Time Limit (seconds)	90
Startup Time Limit (seconds)	90

Name
[name] The application pool name is the unique identifier for the application pool.

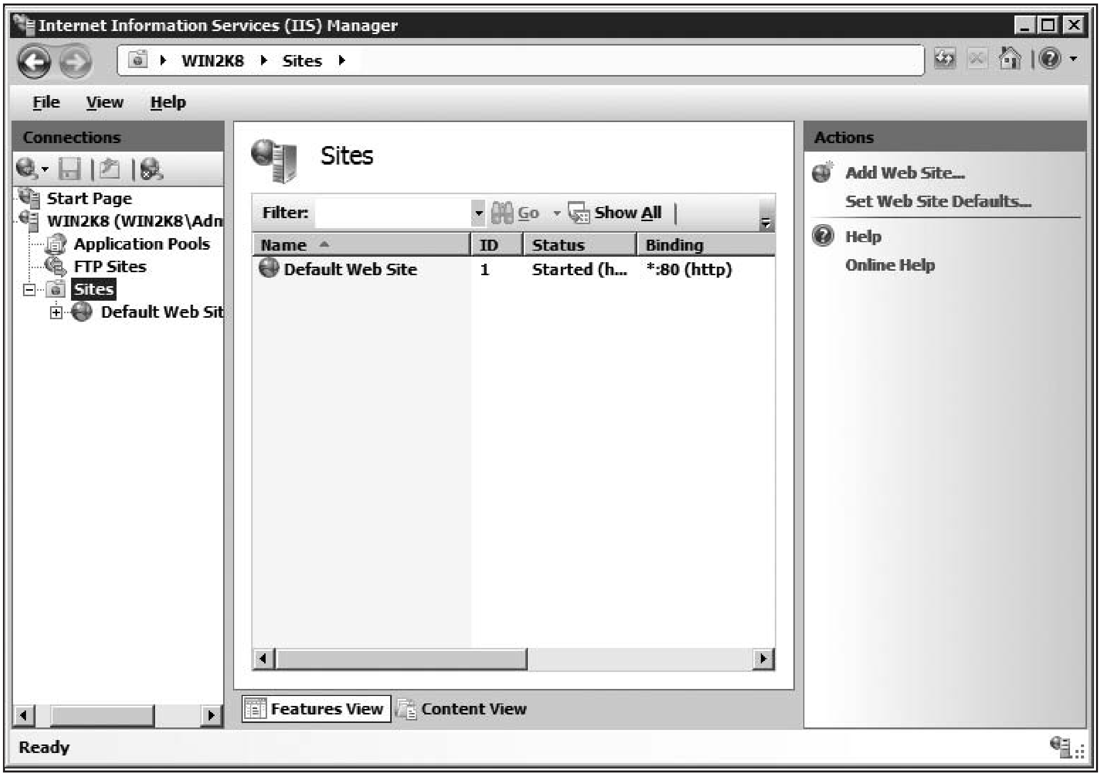
Figure 11-8

Now go to the General section of the Advanced Settings dialog, change the value of the Start Automatically to false (its default of true is shown in Figure 11-8), and click OK. Now if you open the `applicationHost.config` file, you should see the highlighted portion shown in the following code snippet:

```
<system.applicationHost>
  <applicationPools>
    . . .
    <add name="MyApplicationPool" autoStart="false" />
    . . .
  </applicationPools>
</system.applicationHost>
```

In other words, the `applicationHost.config` file records only the values that are different from the default.

Notice that the properties shown in Figure 11-8 map to the XML elements and attributes of the `< applicationPools >` section. When you click the OK button, the callback for this button performs the necessary XML manipulations under the hood to store the changes in the `applicationHost.config` XML file.

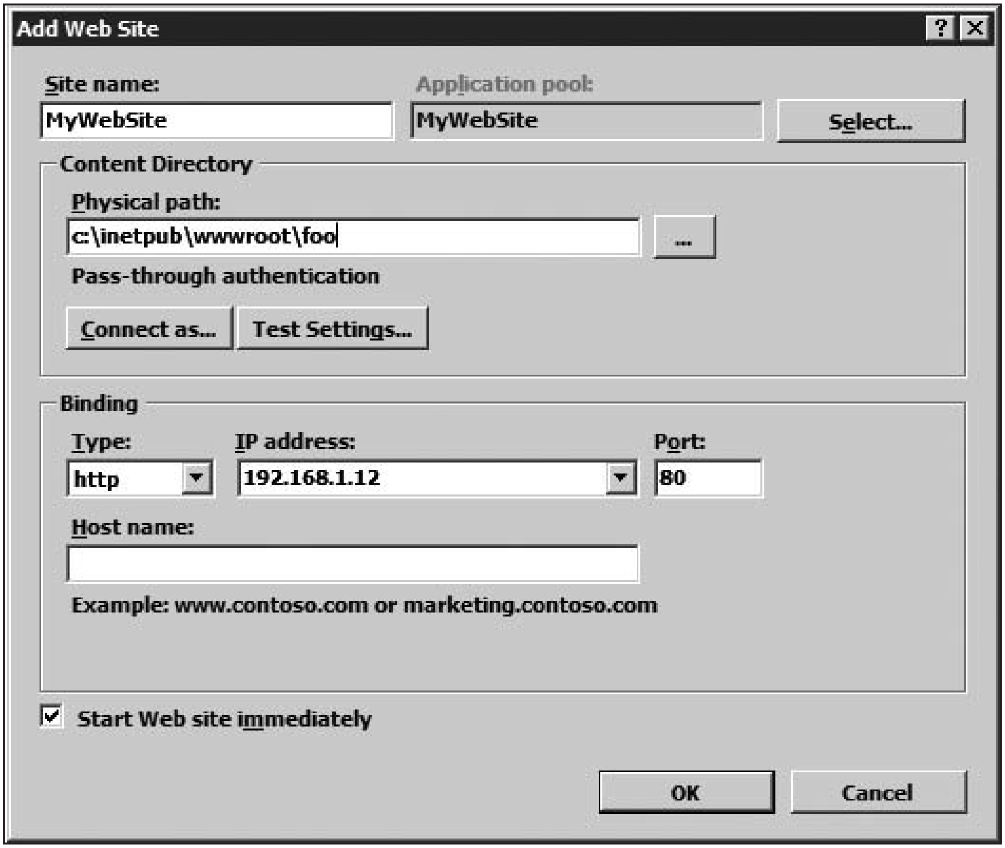

Internet Information Services (IIS) Manager
WIN2K8
Sites
File
View
Help
Connections
Start Page
WIN2K8 (WIN2K8\Adn
Application Pools
FTP Sites
Sites
Default Web Sit
Sites
Filter:
Go
Show All
Features View
Content View
Actions
Add Web Site...
Set Web Site Defaults...
Help
Online Help
Ready

Name	ID	Status	Binding
Default Web Site	1	Started (h...	*:80 (http)

Figure 11-9

Web Sites

Now click the Sites node in the Connections pane of the IIS Manager. In the Actions pane, you should see a link titled Add Web Site, as shown in Figure 11-9. Click the link to launch the dialog shown in Figure 11-10.



Site name: MyWebSite

Application pool: MyWebSite

Physical path: c:\inetpub\wwwroot\foo

Pass-through authentication

Type: http

IP address: 192.168.1.12

Port: 80

Host name:

Example: www.contoso.com or marketing.contoso.com

☑ Start Web site immediately

Figure 11-10

This dialog allows you to add a new Web site. Recall that a Web site is a collection of Web applications. Notice that the properties shown in this dialog map to the XML elements and attributes of the `< site >` element. Next, take these steps:

1. Enter a name in the Web site name text field for the new Web site, for example, MySite.
2. Use the Select button to choose the desired application pool.
3. Choose a physical path.
4. Specify a binding including a binding type, an IP address, and a port number.
5. Click the OK button to commit the changes.

Now open the `applicationHost.config` file again. You should see the highlighted portion shown in Listing 11-3.

Listing 11-3: The applicationHost.config file

```
<configuration>
  <system.applicationHost>
    <sites>
      <site name="MySite" id="1727416169">
        <application path="/">
          <virtualDirectory path="/" physicalPath="C:\inetpub\wwwroot\foo" />
        </application>
        <bindings>
          <binding protocol="http" bindingInformation="192.168.1.12:80:" />
        </bindings>
      </site>
    </sites>
  </system.applicationHost>
</configuration>
```

As Listing 11-3 shows, the dialog shown in Figure 11-10 sets the XML elements and attributes of the `< site >` element that represents the new site. Notice that the dialog automatically created an application with a virtual directory. Every site must have at least one application with the virtual path “/” known as the root application that has at least one virtual directory with the virtual path “/” known as the root virtual directory. This dialog automatically takes care of that requirement behind the scenes.

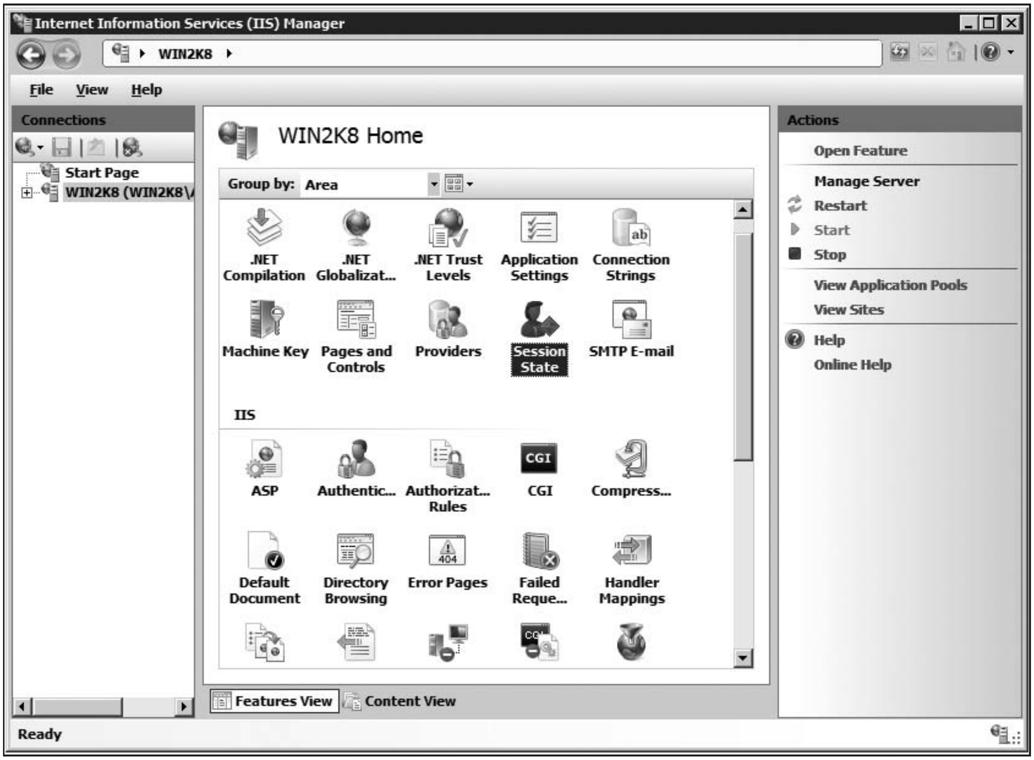

Internet Information Services (IIS) Manager
WIN2K8
File
View
Help
Connections
Start Page
WIN2K8 (WIN2K8\A
WIN2K8 Home
Group by: Area
.NET Compilation
.NET Globalizat...
.NET Trust Levels
Application Settings
Connection Strings
Machine Key
Pages and Controls
Providers
Session State
SMTP E-mail
IIS
ASP
Authentic...
Authorizat... Rules
CGI
Compress...
Default Document
Directory Browsing
Error Pages
Failed Reque...
Handler Mappings
Features View
Content View
Actions
Open Feature
Manage Server
Restart
Start
Stop
View Application Pools
View Sites
Help
Online Help
Ready

Figure 11-11

Hierarchical Configuration

The new IIS7 and ASP.NET 3.5 integrated configuration system consists of a hierarchy of configuration files where lower-level configuration files inherit the configuration settings from higher level configuration files. The lower-level configuration files can override only those inherited configuration settings that are not locked in the higher level configuration files.

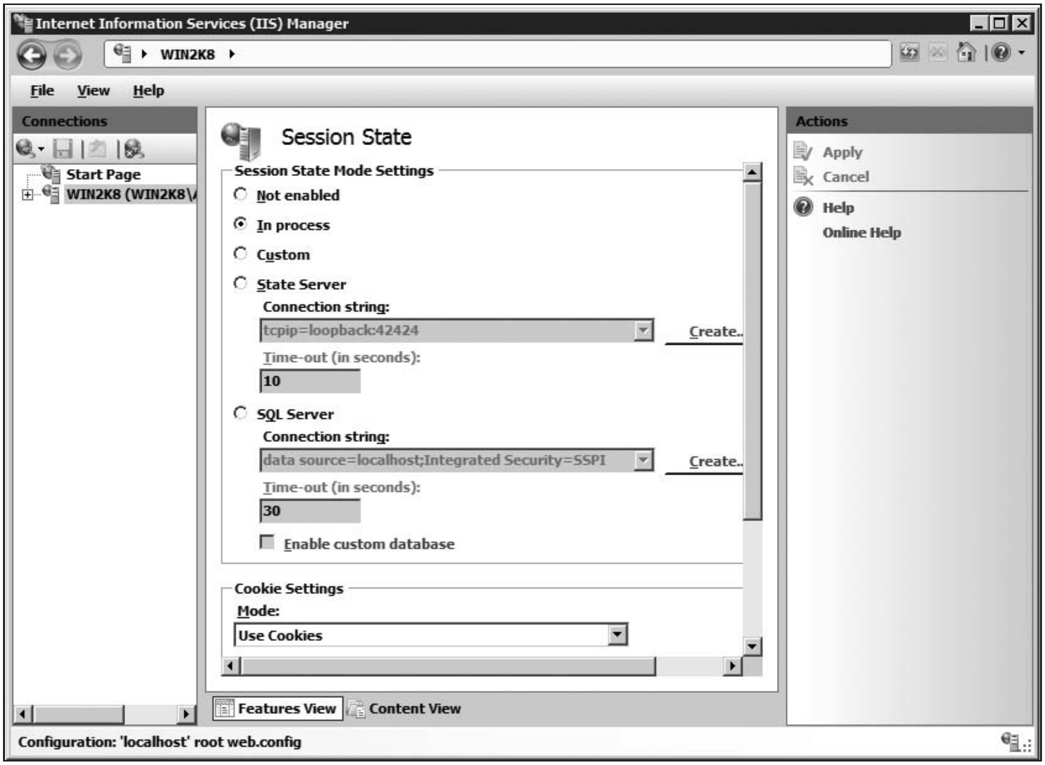
In this section, you'll learn how the IIS Manager takes the hierarchical nature of the IIS7 and ASP.NET 3.5 integrated configuration system into account. Let's begin with the ASP.NET 3.5 configuration settings.

Launch the IIS Manager again, select the node that represents the local Web server in the Connections pane and switch to the Features View tab in the workspace pane. The result should look like Figure 11-11.

Now double-click the Session State icon in the workspace pane. You should see what is shown in Figure 11-12.

Note that the workspace now displays the GUI that allows you to change the session state configuration settings. Go to the Session State Mode Settings section, change the mode to Not enabled, and click the Apply link in the Tasks pane to commit the changes. Now open the root `web.config` file located in the following directory on your machine:

```
%SystemRoot%\Microsoft.NET\Framework\versionNumber\CONFIG\
```


Internet Information Services (IIS) Manager
WIN2K8
File
View
Help
Connections
Start Page
WIN2K8 (WIN2K8\
Session State
Actions
Apply
Cancel
Help
Online Help
Create...
Create...
Features View
Content View

Session State Mode Settings

○ Not enabled

● In process

○ Custom

○ State Server

Connection string: tcpip=loopback:42424

Time-out (in seconds): 10

○ SQL Server

Connection string: data source=localhost;Integrated Security=SSPI

Time-out (in seconds): 30

☐ Enable custom database

Cookie Settings

Mode: Use Cookies

Configuration: 'localhost' root web.config

Figure 11-12

You should see the highlighted portion shown in the following listing:

```
<configuration>
  <system.web>
    <sessionState mode="off" />
  </system.web>
</configuration>
```

As this example shows, you can use IIS Manager to specify the ASP.NET configuration settings as you do for the IIS settings. The tool is smart enough to know that the machine-level ASP.NET configuration settings should be saved into the machine level `web.config` file (known as the root `web.config` file) instead of `applicationHost.config`.

The previous example changed the session state configuration settings at the machine level. Now let's change the session state configuration settings at the site level. Go back to the Connections pane, open the node that represents the local Web server, open the Sites node, and select the Default Web Site node. You should see the result shown in Figure 11-13. *Be sure you select the Web Site node and not the node for the entire Web Server.*

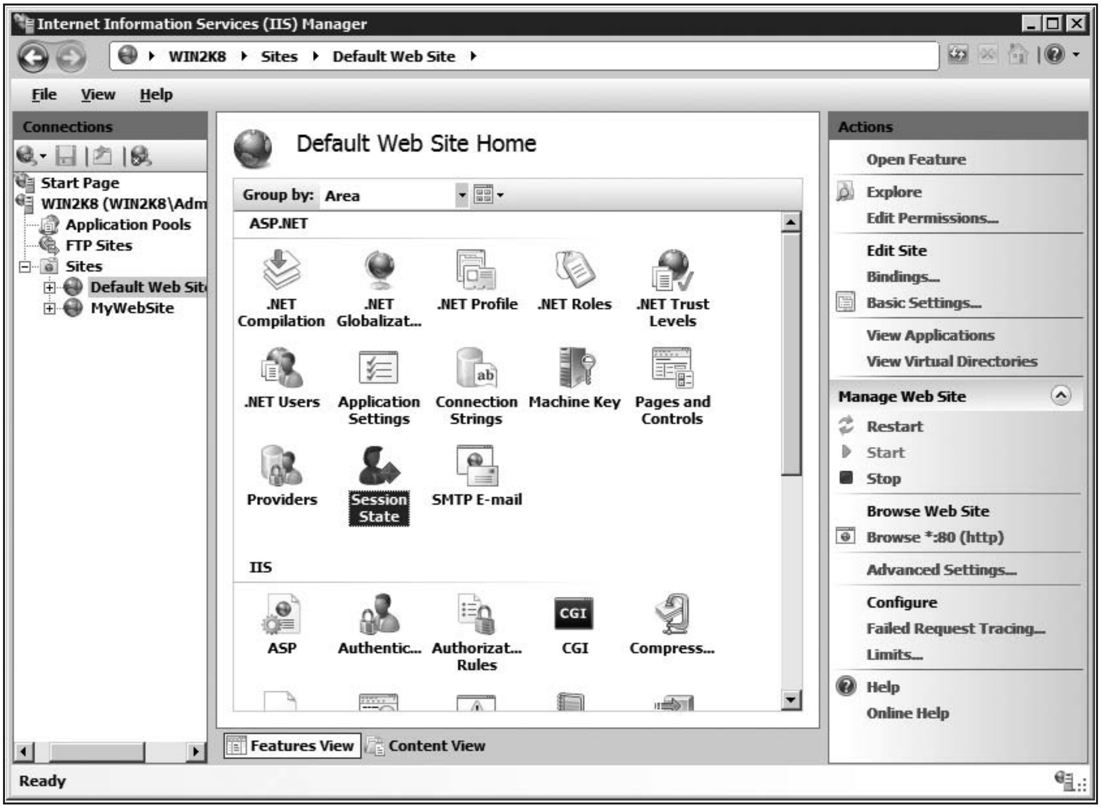

Internet Information Services (IIS) Manager
WIN2K8 ▸ Sites ▸ Default Web Site ▸
File View Help
Connections
Start Page
WIN2K8 (WIN2K8\Adm
Application Pools
FTP Sites
Sites
Default Web Sit
MyWebSite
Default Web Site Home
Group by: Area
ASP.NET
.NET Compilation
.NET Globalizat...
.NET Profile
.NET Roles
.NET Trust Levels
.NET Users
Application Settings
Connection Strings
Machine Key
Pages and Controls
Providers
Session State
SMTP E-mail
IIS
ASP
Authentic...
Authorizat... Rules
CGI
CGI
Compress...
Features View
Content View
Actions
Open Feature
Explore
Edit Permissions...
Edit Site
Bindings...
Basic Settings...
View Applications
View Virtual Directories
Manage Web Site
Restart
Start
Stop
Browse Web Site
Browse *:80 (http)
Advanced Settings...
Configure
Failed Request Tracing...
Limits...
Help
Online Help
Ready

Figure 11-13

Now double-click the Session State icon. Change the Session State Mode settings to Not enabled and click the Apply link on the tasks panel to commit the changes. Now open the `web.config` file in the following directory on your machine:

```
%SystemDrive%\inetpub\wwwroot
```

You should see the highlighted portion of the following code listing:

```
<configuration>
  <system.web>
    <sessionState mode="off" />
  </system.web>
</configuration>
```

As this example shows, the IIS Manager stores the site-level ASP.NET configuration settings to the site-level configuration file. If you repeat the same steps for application level ASP.NET configuration settings, you'll see that the IIS Manager stores these configuration settings into the ASP.NET application level configuration file.

So far you've seen that IIS Manager handles the hierarchical nature of the ASP.NET 3.5 configuration settings. Next, you'll see that the IIS Manager also takes the hierarchical nature of the IIS7 configuration settings into account.

Launch the IIS Manager, click the node that represents the local Web server in the Connections pane, switch to the Features View tab in the workspace, and select the Area option from the Group by combo box to group the items in the workspace by area. You should see the result shown in Figure 11-14.

Now double-click Default Document. The result should look like Figure 11-15.

Notice that the workspace now contains a text box that displays the list of default documents. Add a new default document named **Welcome.htm** to the list and click the Apply button in the task panel to commit the changes.

If you open the `applicationHost.config` file, you should see the highlighted portion shown in Listing 11-4. Notice that the `< files >` element now contains a new `<add>` element whose value attribute is set to `Welcome.html`.

Listing 11-4: The applicationHost.config file

```
<configuration>
  <system.webServer>
    <defaultDocument enabled="true">
      <files>
        <clear />
        <add value="Welcome.htm" />
        <add value="Default.asp" />
        <add value="index.htm" />
        <add value="index.html" />
        <add value="iisstart.htm" />
        <add value="default.aspx" />
      </files>
    </defaultDocument>
  </system.webServer>
</configuration>
```

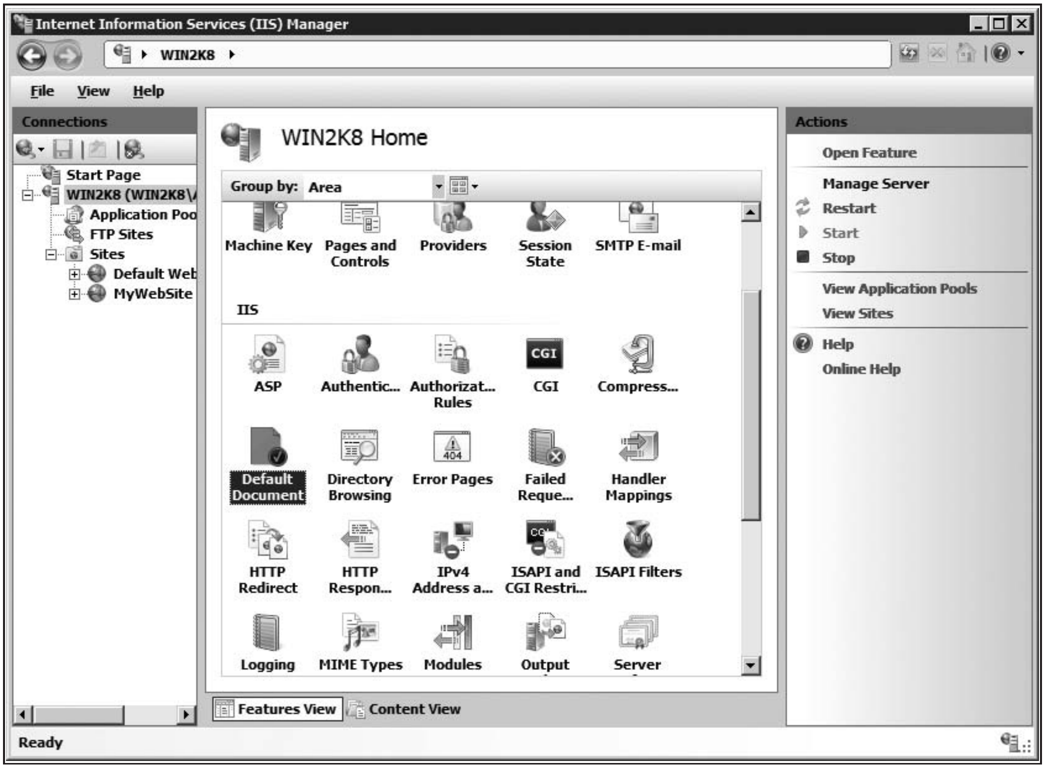
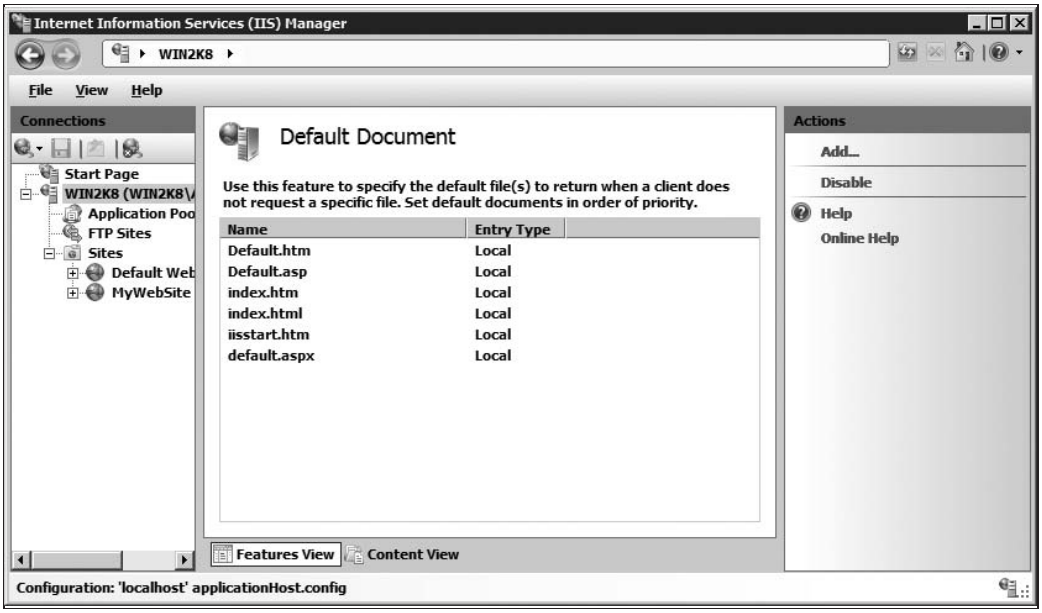

Internet Information Services (IIS) Manager
WIN2K8
File View Help
Connections
Start Page
WIN2K8 (WIN2K8\A
Application Poo
FTP Sites
Sites
Default Web
MyWebSite
WIN2K8 Home
Group by: Area
Machine Key
Pages and Controls
Providers
Session State
SMTP E-mail
IIS
ASP
Authentic...
Authorizat... Rules
CGI
Compress...
Default Document
Directory Browsing
Error Pages
Failed Reque...
Handler Mappings
HTTP Redirect
HTTP Respon...
IPv4 Address a...
ISAPI and CGI Restri...
ISAPI Filters
Logging
MIME Types
Modules
Output
Server
Features View
Content View
Actions
Open Feature
Manage Server
Restart
Start
Stop
View Application Pools
View Sites
Help
Online Help
Ready

Figure 11-14


Internet Information Services (IIS) Manager
WIN2K8
File View Help
Connections
Start Page
WIN2K8 (WIN2K8\A
Application Poo
FTP Sites
Sites
Default Web
MyWebSite
Default Document
Features View
Content View
Actions
Add...
Disable
Help
Online Help

Use this feature to specify the default file(s) to return when a client does not request a specific file. Set default documents in order of priority.

Name	Entry Type
Default.htm	Local
Default.asp	Local
index.htm	Local
index.html	Local
iisstart.htm	Local
default.aspx	Local

Configuration: 'localhost' applicationHost.config

Figure 11-15

Now select the Default Web Site node from the Connections pane of the IIS Manager and double-click the Default Document icon to go to the page that displays the list of default documents. Note that the list contains the `Welcome.html` default document that you added before. This makes sense because the Default Web Site inherits all the default documents settings from the machine-level `applicationHost.config` file. Now go ahead and remove the `Welcome.html` file from the list, add a new default document named **Start.html**, and click the Apply button to commit the changes. If you open the `web.config` file located in the Default Web Site's root directory, you should see the result shown in Listing 11-5.

Listing 11-5: The root web.config file

```
<configuration>
  <system.webServer>
    <defaultDocument>
      <files>
        <clear />
        <add value="Start.htm" />
        <add value="Default.asp" />
        <add value="index.htm" />
        <add value="index.html" />
        <add value="iisstart.htm" />
        <add value="default.aspx" />
      </files>
    </defaultDocument>
  </system.webServer>
</configuration>
```

Delegation

As Listing 11-5 shows, a site- or application-level `web.config` file now can contain both ASP.NET and IIS configuration sections. This is one of the great new features of IIS7, which provides the following two benefits among many others:

- It allows you to configure IIS7 to meet your application-specific requirements.
- Since these IIS7 custom configuration settings are all stored in the `web.config` file of your application, which is located in the same directory with the rest of your application, you can xcopy this configuration file together with the rest of your application to the test machine, and from there to the production machine. This will allow your testers and clients to start testing and using your applications right away, without going through the tedious task of reconfiguring their Web servers to match the configuration you had on your Web server when you were developing the application.

You may be wondering whether it is a good idea to allow site and application administrators or developers to mess with the Web server settings from a security perspective. IIS7 and ASP.NET 3.5 integrated configuration system has taken this security issue into account. Because of the hierarchical nature of the configuration files, changes made to a configuration file at a certain level of hierarchy apply only to that level and the levels below it. For example, if you make some configuration changes in the `web.config` located in the root directory of a site, it will affect only the applications and virtual directories in that site. Or if you make changes in the `web.config` located in the root directory of an application, it will affect only the virtual directories in that application.

In addition most IIS configuration sections are locked by default at installation, which means that by default only the machine administrator can change these locked IIS configuration sections. However, the machine administrator can remove the lock from selected IIS configuration sections to allow selected sites, applications, or virtual directories to change these configuration sections. This is known as delegation. Let's take a look at an example.

Recall from the previous example (see Listing 11-5) that the Default Web Site administrator was allowed to reconfigure the IIS7 default documents for all the applications and virtual directories of the Default Web Site. This was possible because by default there is no lock on the IIS7 default documents feature. To see this, take the following steps:

1. Launch the IIS Manager.
2. Click the node that represents the local Web server in the Connections pane.
3. Switch to the Features View tab in the workspace.
4. Select the Area option from the Group by combo box of the workspace.

The result should look like Figure 11-16.

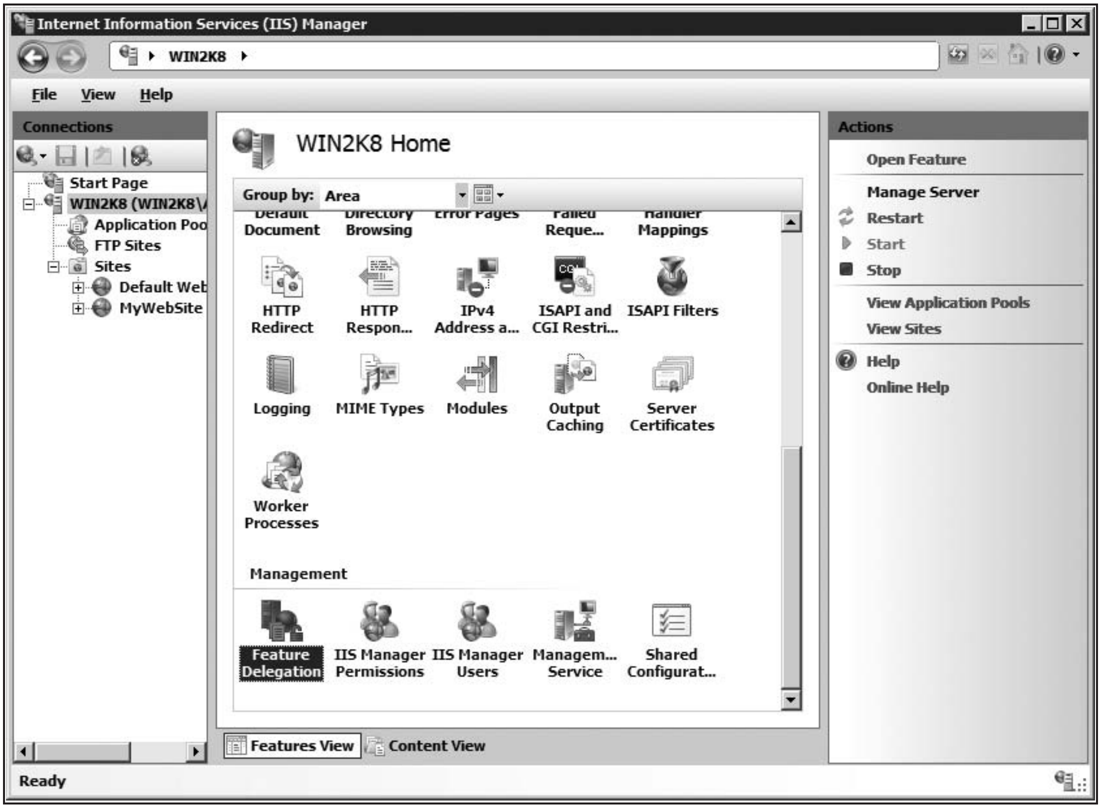
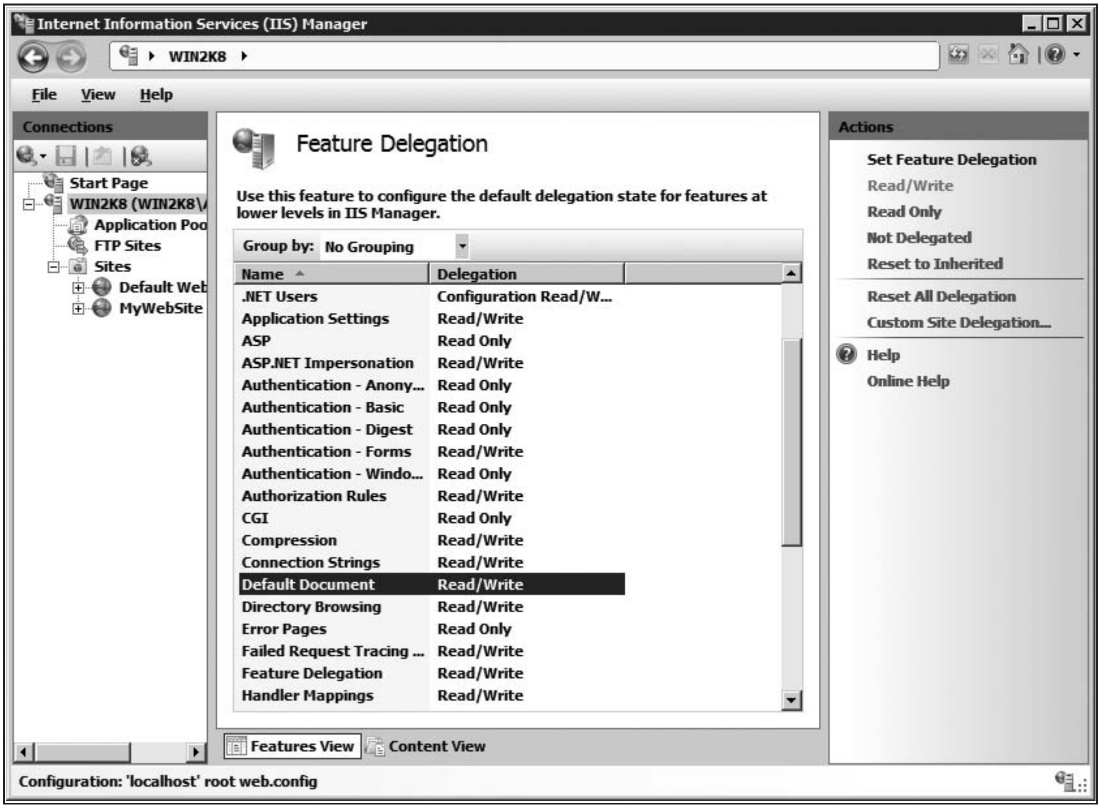

Internet Information Services (IIS) Manager
WIN2K8
File
View
Help
Connections
Start Page
WIN2K8 (WIN2K8\
Application Poo
FTP Sites
Sites
Default Web
MyWebSite
WIN2K8 Home
Group by: Area
Default Document
Directory Browsing
Error Pages
Failed Reque...
Handler Mappings
HTTP Redirect
HTTP Respon...
IPv4 Address a...
ISAPI and CGI Restri...
ISAPI Filters
Logging
MIME Types
Modules
Output Caching
Server Certificates
Worker Processes
Management
Feature Delegation
IIS Manager Permissions
IIS Manager Users
Managem... Service
Shared Configurat...
Features View
Content View
Actions
Open Feature
Manage Server
Restart
Start
Stop
View Application Pools
View Sites
Help
Online Help
Ready

Figure 11-16

Now double-click the Feature Delegation icon in the Management section of the workspace to go to the Feature Delegation page shown in Figure 11-17. As the name implies, the Feature Delegation page allows the machine administrator to delegate the administration of the selected IIS features to site and applications administrators. Select the Delegation option from the Group by combo box and go to the Read/Write section of this page as shown in Figure 11-17. As the title of this section implies, this section contains IIS7 features that can be read and written from the lower-level configuration files. Note that this section contains the Default Document feature.


Internet Information Services (IIS) Manager
WIN2K8
File
View
Help
Connections
Start Page
WIN2K8 (WIN2K8\
Application Poo
FTP Sites
Sites
Default Web
MyWebSite
Feature Delegation
Actions
Set Feature Delegation
Read/Write
Read Only
Not Delegated
Reset to Inherited
Reset All Delegation
Custom Site Delegation...
Help
Online Help
Features View
Content View

Use this feature to configure the default delegation state for features at lower levels in IIS Manager.

Group by: No Grouping

Name	Delegation
.NET Users	Configuration Read/W...
Application Settings	Read/Write
ASP	Read Only
ASP.NET Impersonation	Read/Write
Authentication - Anony...	Read Only
Authentication - Basic	Read Only
Authentication - Digest	Read Only
Authentication - Forms	Read/Write
Authentication - Windo...	Read Only
Authorization Rules	Read/Write
CGI	Read Only
Compression	Read/Write
Connection Strings	Read/Write
Default Document	Read/Write
Directory Browsing	Read/Write
Error Pages	Read Only
Failed Request Tracing ...	Read/Write
Feature Delegation	Read/Write
Handler Mappings	Read/Write

Configuration: 'localhost' root web.config

Figure 11-17

Select the Default Document from the Feature Delegation page as shown in Figure 11-17. Notice that the task pane contains a section titled Set Feature Delegation. This section contains six links named:

- ❑ Read/Write
- ❑ Read Only
- ❑ Not Delegated (on Windows Server 2008) or Remove Delegation (on Windows Vista)
- ❑ Reset to inherited
- ❑ Reset All Delegation
- ❑ Custom Web Site Delegation (This link only exists on Windows Server 2008.)

Note that the Read/Write link in the Actions Pane is grayed out (as is the content-menu item), which means that the lower-level configuration files have the permission to change the IIS7 Default

Document feature. Click the Read Only link and open the `applicationHost.config` file. You should see the highlighted portion shown in Listing 11-6. Some parts have been omitted for brevity.

Listing 11-6: The applicationHost.config file

```
<configuration>
  . . .
   <location path="" overrideMode="Deny">
     <system.webServer>
       <defaultDocument enabled="true">
         <files>
           <clear />
           <add value="Welcome.htm" />
           <add value="Default.asp" />
           <add value="index.htm" />
           <add value="index.html" />
           <add value="iisstart.htm" />
           <add value="default.aspx" />
         </files>
       </defaultDocument>
     </system.webServer>
   </location>
   . . .
 </configuration>
```

As Listing 11-6 shows, the IIS Manager has added a new `< location >` tag whose `overrideMode` attribute is set to `Deny` to signal that the machine administrator does not want any lower-level configuration file to change the IIS7 default document feature. This means that every site, application, and virtual directory running on the machine inherits these authorization rules and has to live by them.

Moving an Application from IIS6 to IIS7

If you add a standard ASP.NET application into an IIS application pool that is configured for the Integrated Pipeline, rather than the Classic Pipeline and it contains entries in its `web.config` for `< system.web > / < httpModules >`, you'll get an informative error message like the one in Figure 11-18.

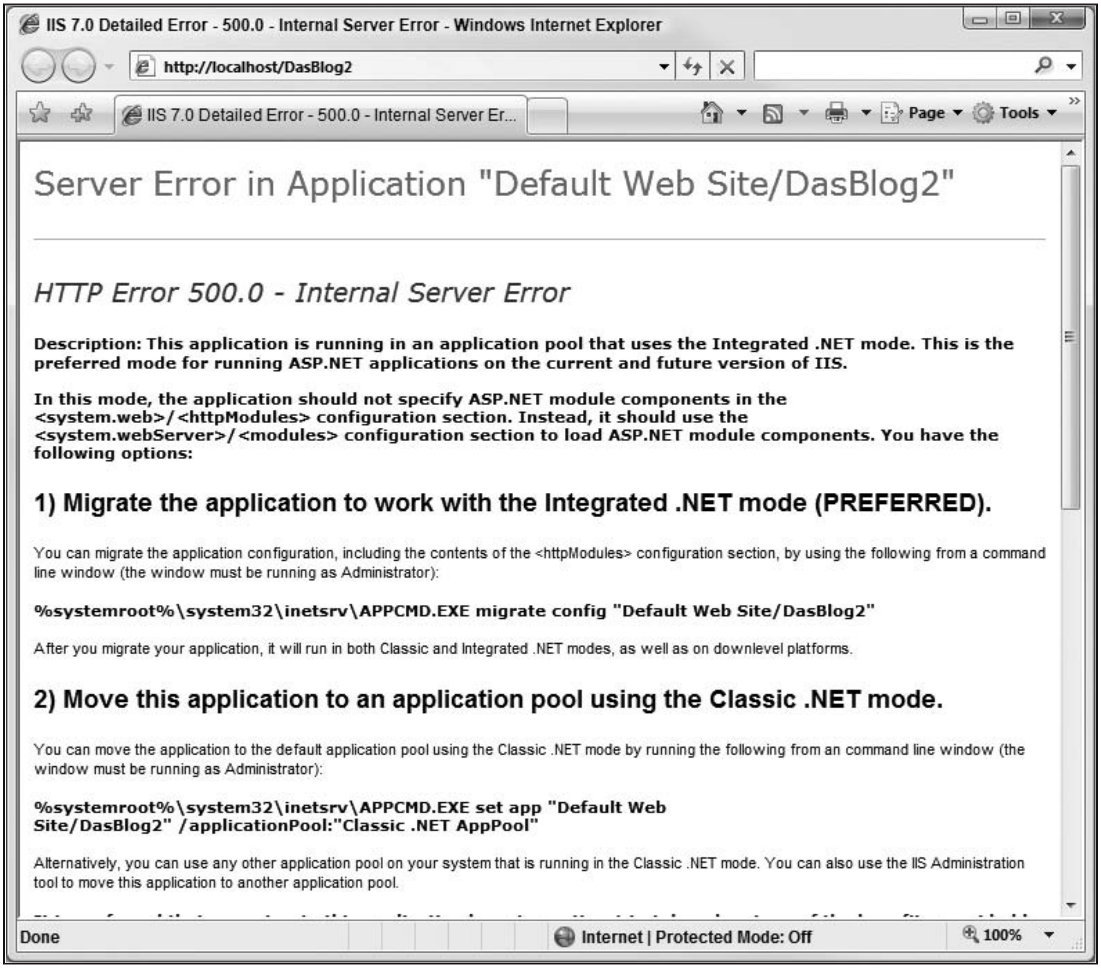
At this point you have two choices, as clearly outlined in the error message. You can either change the application's `web.config` to move the modules into the IIS7 integrated pipeline, or you can run the application in Classic mode.

The error message actually includes the command line you need to migrate your `web.config`:

```
%systemroot%\system32\inetsrv\APPCMD.EXE migrate config "Default Web Site/DasBlog2"
```

When you run that statement from an administrator command line, you'll see the following output:

```
%systemroot%\system32\inetsrv\APPCMD.EXE migrate config "Default Web Site/DasBlog2"
Successfully migrated section "system.web/httpModules".
Successfully migrated section "system.web/httpHandlers".
```


IIS 7.0 Detailed Error - 500.0 - Internal Server Error - Windows Internet Explorer
http://localhost/DasBlog2
IIS 7.0 Detailed Error - 500.0 - Internal Server Er...
Page
Tools
Done
Internet | Protected Mode: Off
100%

Server Error in Application "Default Web Site/DasBlog2"

HTTP Error 500.0 - Internal Server Error

Description: This application is running in an application pool that uses the Integrated .NET mode. This is the preferred mode for running ASP.NET applications on the current and future version of IIS.

In this mode, the application should not specify ASP.NET module components in the <system.web>/<httpModules> configuration section. Instead, it should use the <system.webServer>/<modules> configuration section to load ASP.NET module components. You have the following options:

1) Migrate the application to work with the Integrated .NET mode (PREFERRED).

You can migrate the application configuration, including the contents of the <httpModules> configuration section, by using the following from a command line window (the window must be running as Administrator):

%systemroot%\system32\inetsrv\APPCMD.EXE migrate config "Default Web Site/DasBlog2"

After you migrate your application, it will run in both Classic and Integrated .NET modes, as well as on downlevel platforms.

2) Move this application to an application pool using the Classic .NET mode.

You can move the application to the default application pool using the Classic .NET mode by running the following from an command line window (the window must be running as Administrator):

%systemroot%\system32\inetsrv\APPCMD.EXE set app "Default Web Site/DasBlog2" /applicationPool:"Classic .NET AppPool"

Alternatively, you can use any other application pool on your system that is running in the Classic .NET mode. You can also use the IIS Administration tool to move this application to another application pool.

Figure 11-18

Be sure to make a copy of your `web.config` for backup purposes, but also for education as you can compare the two with your favorite diff tool. Notice the creation of a new `system.webServer` section in the following snippet:

```
      <system.webServer>
          <modules>
              <add name="UrlMapperModule"
                type="newtelligence.DasBlog.Web.Core.UrlMapperModule,
                newtelligence.DasBlog.Web.Core" preCondition="managedHandler" />
  ...removed for brevity...
          </modules>
          <handlers>
              <add name="*.blogtemplate_*" path="*.blogtemplate" verb="*"
                type="System.Web.HttpForbiddenHandler"
                preCondition="integratedMode,runtimeVersionv2.0" />
```

```
...removed for brevity...
        </handlers>
    </system.webServer>
```

Notice also the `preCondition` attribute that was automatically created by the migration tool for both the handlers and the modules. When it is set to `managedHandler`, this means that by default all registered managed modules will be applied only to those requests whose handlers are managed handlers, that is, requests for ASP.NET content.

Summary

IIS7 is easier to install than IIS6 was. When you choose only those modules required for your application, you create a more secure Web Server. IIS7 is easier to configure than IIS6. There's a much richer graphical interface, and command-line interface, and a transparent and hierarchical XML-based configuration system that feels familiar to professional ASP.NET developers. Applications are trivial to migrate and run faster under IIS7.

The skills you've developed writing HttpHandlers and HttpModules can be immediately leveraged within IIS7. You should check out IIS7 on Vista or Windows Server 2008 and pick up *Professional IIS 7 and ASP.NET Integrated Programming* from Wrox for an in-depth look at the IIS 7 programming model.

12

Introduction to the Provider Model

The ASP.NET provider model is an important framework to understand as you build your applications. The ASP.NET provider model was a major change introduced in ASP.NET 2.0, so you probably already know that ASP.NET is a way to build applications for the Internet. *For the Internet* means that an application's display code travels over HTTP, which is a stateless protocol. ASP.NET works with a disconnected architecture. The simple nature of this model means that requests come in and then responses are sent back. On top of that, ASP.NET does not differentiate one request from another. The server containing an ASP.NET application is simply reacting to any request thrown at it.

This means that a developer building a Web application has to put some thought into how users can remain in context between their requests to the server as they work through the application. Keeping a user in context means recording state, the state of the user, to some type of data store. This can be done in multiple ways, and no one way is the *perfect* way. Rather, you have to choose one of the available methods.

> *You can read about maintaining state in an ASP.NET application in Chapter 22.*

State can be stored via multiple methods, some of which include:

- ❑ Application State
- ❑ Session State
- ❑ The Cache Object

You use all these methods on the server, but you can also employ your own custom methods — such as simply storing state in a database using your own custom schema. It is also possible to write state back to the clients, either directly on their computers or by placing state in the HTML output in the response. Some of these methods include:

- ❑ Cookies
- ❑ Querystrings
- ❑ Hidden Fields
- ❑ ViewState

Understanding the Provider

These methods work rather well; but most of them are rudimentary, and have short life spans. ASP.NET 3.5 includes a handful of systems (such as the new membership and role management systems) that handle state for users between multiple requests/response transactions. In fact, these systems require state management capabilities that go well beyond the limited time frames that are possible in the previously mentioned state management methods. Therefore, many of these systems must record state in more advanced modes — something that is easy to do in ASP.NET 3.5. Recording state to data stores in more advanced modes is accomplished through the use of *providers*.

> **A *provider* is an object that allows for programmatic access to data stores, processes, and more.**

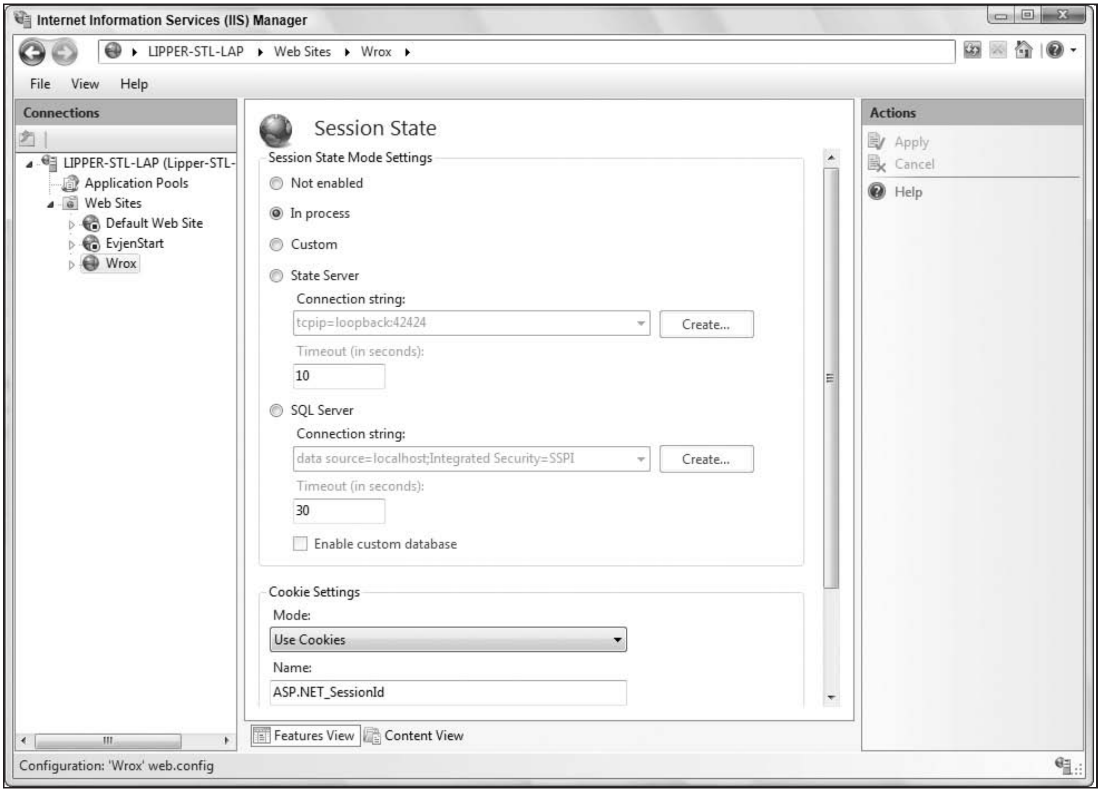
When working with ASP.NET 1.*x*, you might have encountered a rudimentary provider model that was present in the system. This provider model was an object that sat between the `Session` object and the actual place in which the sessions were stored. By default, sessions in ASP.NET are stored `InProc`, meaning in the same process where ASP.NET is running. In ASP.NET 1.*x* (and in ASP.NET 3.5 for that matter), you can simply change the provider used for the `Session` object; this will, in turn, change where the session is stored. The available providers for storing session information include:

- ❑ `InProc`
- ❑ `StateServer`
- ❑ `SQLServer`

Besides `InProc`, you can use `StateServer` that enables you to store sessions in a process that is entirely separate from the one in which ASP.NET runs. This protects your sessions if the ASP.NET process shuts down. You can also store your sessions to disk (in a database for instance) using the `SQLServer` option. This method enables you to store your sessions directly in Microsoft's SQL Server. How do you go about changing the provider that is used for sessions? You can do this in a couple of ways.

One option to change the provider used for sessions is through the Internet Information Services (IIS) Manager, as shown in Figure 12-1.

The other option is to go directly to a system-wide configuration file (such as the `machine.config` file) or to an application configuration file (such as the `web.config`). In the file, change the name of the session state provider that is to be used within the `<sessionState>` section of the configuration document.


Internet Information Services (IIS) Manager
LIPPER-STL-LAP ▸ Web Sites ▸ Wrox ▸
File View Help
Connections
LIPPER-STL-LAP (Lipper-STL-
Application Pools
Web Sites
Default Web Site
EvjenStart
Wrox
Actions
Apply
Cancel
Help
Create...
Create...
Features View
Content View

Session State

Session State Mode Settings

○ Not enabled

● In process

○ Custom

○ State Server

Connection string: tcpip=loopback:42424

Timeout (in seconds): 10

○ SQL Server

Connection string: data source=localhost;Integrated Security=SSPI

Timeout (in seconds): 30

☐ Enable custom database

Cookie Settings

Mode: Use Cookies

Name: ASP.NET_SessionId

Configuration: 'Wrox' web.config

Figure 12-1

Since ASP.NET 2.0, you have been able to take this provider model one step further than you ever could before. You discover this next.

The Provider Model in ASP.NET 3.5

Back when ASP.NET 2.0 was being developed, plenty of requests came into the ASP.NET team. Users wanted to be able to store sessions by means other than the three methods — `InProc`, `StateServer`, and `SQLServer`. For instance, one such request was for a provider that could store sessions in an Oracle database. This might seem as if it's a logical thing to add to ASP.NET in the days of ASP.NET 1.1. But if the team added a provider for Oracle, they would soon get requests to add even more providers for other databases and data storage methods. For this reason, instead of building providers for each and every possible scenario, the developers designed a provider model that enabled them to add any providers they wished. Thus, the new provider model found in ASP.NET was born.

ASP.NET 3.5 includes a lot of systems that required state storage of some kind. Also, instead of recording state in a fragile mode (the way sessions are stored by default), many of these systems require that their state be stored in more concrete data stores such as databases or XML files. This also allows a longer-lived state for the users visiting an application — something else that is required by these systems.

The systems based upon the provider model found in ASP.NET 3.5 that require advanced state management include the following:

- ❑ Membership
- ❑ Role management
- ❑ Site navigation
- ❑ Personalization
- ❑ Health monitoring Web events
- ❑ Web parts personalization
- ❑ Configuration file protection

The membership system is a means to allow ASP.NET to work from a user store of some kind to create, delete, or edit application users. Because it is rather apparent that developers want to work with an unlimited amount of different data stores for their user store, they need a means to change the underlying user store for their ASP.NET applications easily. The provider model found in ASP.NET 3.5 is the answer.

Out of the box, ASP.NET 3.5 provides a couple of membership providers that enable you to store user information. The included providers are the SQL Server and the Active Directory membership providers (found at `System.Web.Security.SqlMembershipProvider` and `System.Web.Security.ActiveDirectoryMembershipProvider`, respectively). In fact, for each of the systems (as well as for some of the ASP.NET 1.*x* systems), a series of providers is available to alter the way the state of that system is recorded. Figure 12-2 illustrates these new providers.

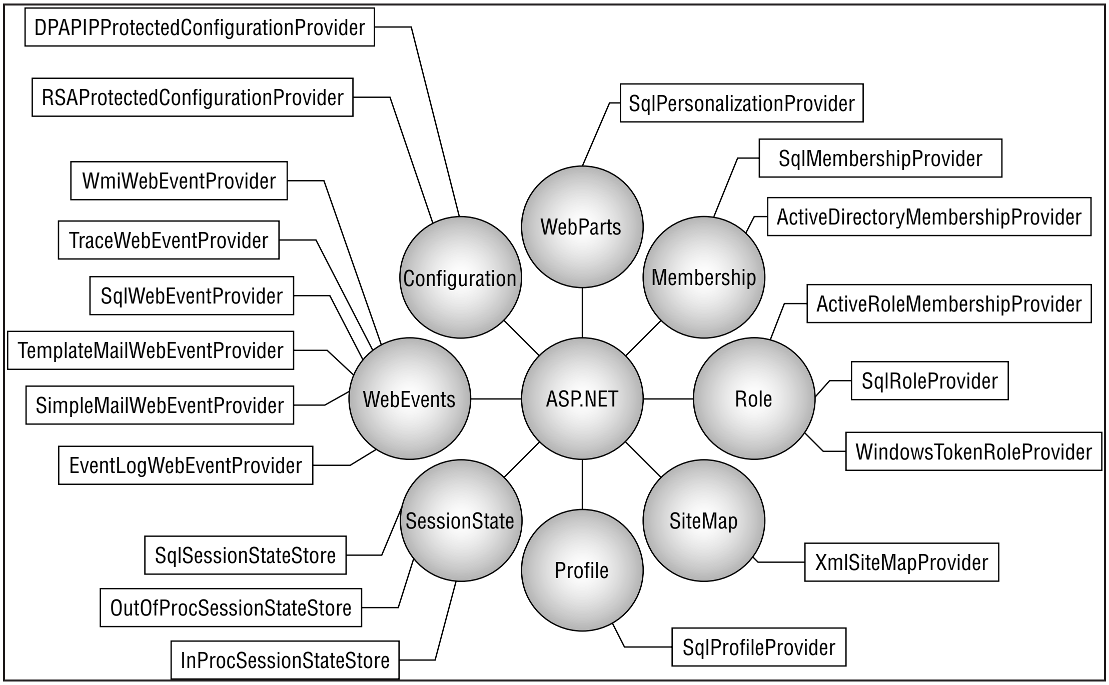

DPAPIPProtectedConfigurationProvider
RSAProtectedConfigurationProvider
SqlPersonalizationProvider
SqlMembershipProvider
WmiWebEventProvider
ActiveDirectoryMembershipProvider
TraceWebEventProvider
WebParts
Configuration
Membership
SqlWebEventProvider
ActiveRoleMembershipProvider
TemplateMailWebEventProvider
SqlRoleProvider
SimpleMailWebEventProvider
WebEvents
ASP.NET
Role
WindowsTokenRoleProvider
EventLogWebEventProvider
SessionState
SiteMap
SqlSessionStateStore
XmlSiteMapProvider
Profile
OutOfProcSessionStateStore
SqlProfileProvider
InProcSessionStateStore

Figure 12-2

As you can see from the diagram, ASP.NET provides a large number of providers out of the box. Some systems have only a single provider (such as the profile system that includes only a provider to connect to SQL Server), whereas other systems include multiple providers (such as the WebEvents provider that includes six separate providers). Next, this chapter reviews how to set up SQL Server to work with a number of the providers presented in this chapter. You can use SQL Server 7.0, 2000, 2005, or 2008 for the backend data store for many of the providers presented (although not all of them). After this explanation, you review each of the available providers built into ASP.NET 3.5.

Setting Up Your Provider to Work with Microsoft SQL Server 7.0, 2000, 2005, or 2008

Quite a number of providers work with SQL Server. For instance, the membership, role management, personalization, and other systems work with SQL Server right out of the box. However, all these systems work with the new Microsoft SQL Server Express Edition file (`.mdf`) by default instead of with one of the full-blown versions of SQL Server such as SQL Server 7.0, SQL Server 2000, SQL Server 2005, or SQL Server 2008.

To work with either of these databases, you must set up the database using the `aspnet_regsql.exe` tool. Working with `aspnet_regsql.exe` creates the necessary tables, roles, stored procedures, and other items needed by the providers. To get at this tool, open up the Visual Studio 2008 Command Prompt by selecting Start ➪All Programs ➪Microsoft Visual Studio 2008 ➪Visual Studio Tools ➪Visual Studio 2008 Command Prompt. This gives you access to the ASP.NET SQL Server Setup Wizard. The ASP.NET SQL Server Setup Wizard is an easy-to-use tool that facilitates setup of the SQL Server to work with many of the systems that are built into ASP.NET 3.5, such as the membership, role management, and personalization systems. The Setup Wizard provides two ways for you to set up the database: using a command-line tool or using a GUI tool. First, look at the command-line version of the tool.

The ASP.NET SQL Server Setup Wizard Command-Line Tool

The command-line version of the Setup Wizard gives the developer optimal control over how the database is created. Working from the command-line using this tool is not difficult, so don't be intimidated by it.

You can get at the actual tool, `aspnet_regsql.exe`, from the Visual Studio Command Prompt if you have Visual Studio 2008. At the command prompt, type `aspnet_regsql.exe -?` to get a list of all the command-line options at your disposal for working this tool.

The following table describes some of the available options for setting up your SQL Server instance to work with the personalization framework.

Command Option	Description
`-?`	Displays a list of available option commands.
`-W`	Uses the Wizard mode. This uses the default installation if no other parameters are used.
`-S <server>`	Specifies the SQL Server instance to work with.
`-U <login>`	Specifies the username to log in to SQL Server. If you use this, you also use the `-P` command.

Command Option	Description
`-P <password>`	Specifies the password to use for logging in to SQL Server. If you use this, you also use the `-U` command.
`-E`	Provides instructions to use the current Windows credentials for authentication.
`-C`	Specifies the connection string for connecting to SQL Server. If you use this, you can avoid using the `-U` and `-P` commands because they are specified in the connection string itself.
`-A all`	Adds support for all the available SQL Server operations provided by ASP.NET 3.5 including membership, role management, profiles, site counters, and page/control personalization.
`-A p`	Adds support for working with profiles.
`_R all`	Removes support for all the available SQL Server operations that have been previously installed. These include membership, role management, profiles, site counters, and page/control personalization.
`-R p`	Removes support for the profile capability from SQL Server.
`-d <database>`	Specifies the database name to use with the application services. If you don't specify a name of a database, `aspnetdb` is used.
`/sqlexportonly <filename>`	Instead of modifying an instance of a SQL Server database, use this command in conjunction with the other commands to generate a SQL script that adds or removes the features specified. This command creates the scripts in a file that has the name specified in the command.

To modify SQL Server to work with the personalization provider using this command-line tool, you enter a command such as the following:

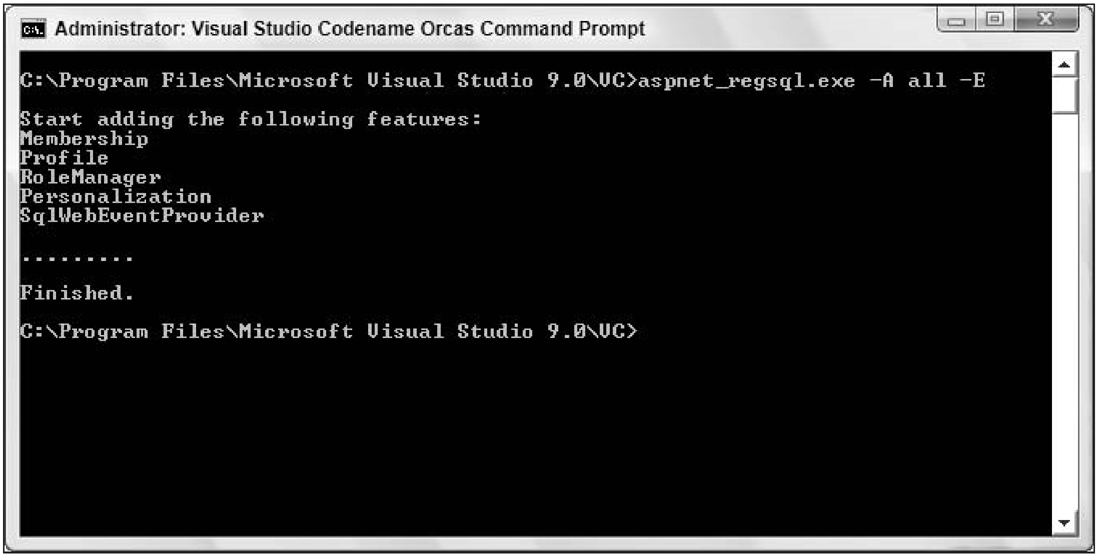
```
aspnet_regsql.exe -A all -E
```

After you enter the preceding command, the command-line tool creates the features required by all the available ASP.NET 3.5 systems. The results are shown in the tool itself, as you see in Figure 12-3.

When this action is completed, you can see that a new database, `aspnetdb`, has been created in the Microsoft SQL Server Management Studio, which is part of Microsoft SQL Server 2005 (the database used for this example). You now have the appropriate tables for working with all the ASP.NET 3.5 systems that are able to work with SQL Server (see Figure 12-4).

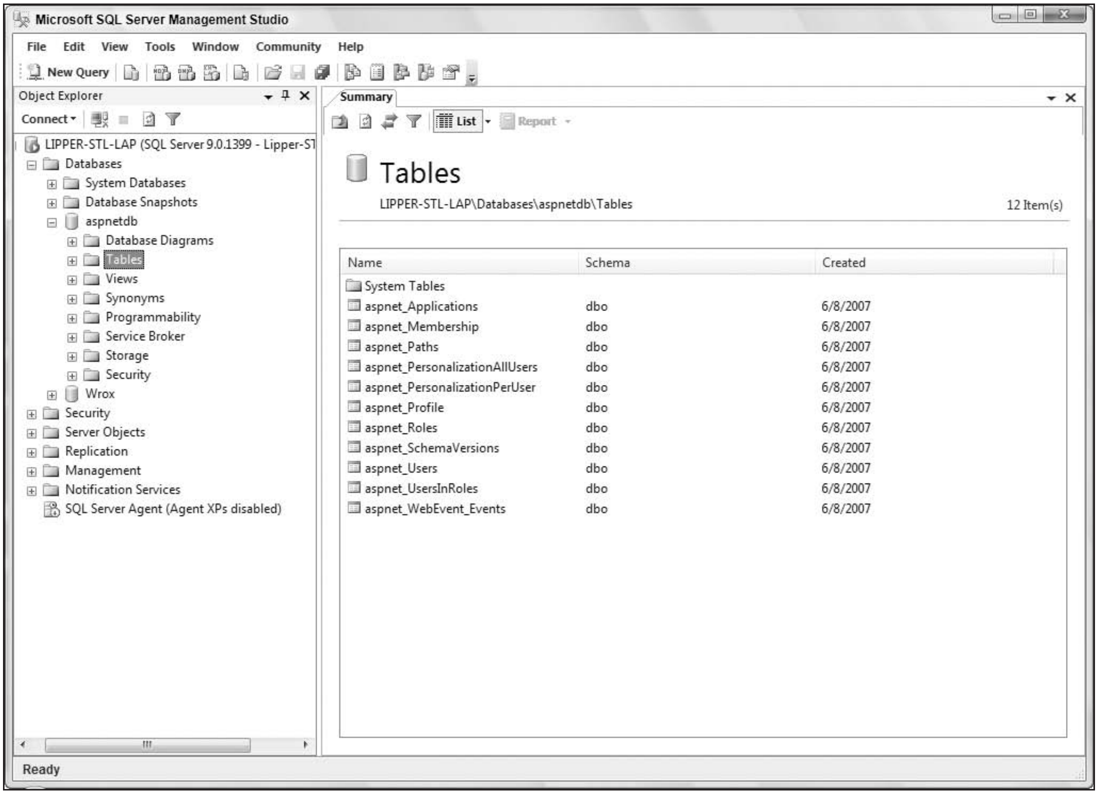
One advantage of using the command-line tool rather than the GUI-based version of the ASP.NET SQL Server Setup Wizard is that you can install in the database just the features that you are interested in working with instead of installing everything (as the GUI-based version does). For instance, if you are going to have only the membership system interact with SQL Server 2005 — not any of the other systems (such as role management and personalization) — then you can configure the setup so that only the tables, roles, stored procedures, and other items required by the membership system are established in the database. To set up the database for the membership system only, you use the following command on the command line.

```
aspnet_regsql.exe -A m -E
```


Administrator: Visual Studio Codename Orcas Command Prompt

```
C:\Program Files\Microsoft Visual Studio 9.0\VC>aspnet_regsql.exe -A all -E

Start adding the following features:
Membership
Profile
RoleManager
Personalization
SqlWebEventProvider

.........

Finished.

C:\Program Files\Microsoft Visual Studio 9.0\VC>
```

Figure 12-3


Microsoft SQL Server Management Studio
File
Edit
View
Tools
Window
Community
Help
New Query
Object Explorer
Connect
LIPPER-STL-LAP (SQL Server 9.0.1399 - Lipper-ST
Databases
System Databases
Database Snapshots
aspnetdb
Database Diagrams
Tables
Views
Synonyms
Programmability
Service Broker
Storage
Security
Wrox
Security
Server Objects
Replication
Management
Notification Services
SQL Server Agent (Agent XPs disabled)
Summary
List
Report
Tables
LIPPER-STL-LAP\Databases\aspnetdb\Tables
12 Item(s)
Ready

Name	Schema	Created
System Tables		
aspnet_Applications	dbo	6/8/2007
aspnet_Membership	dbo	6/8/2007
aspnet_Paths	dbo	6/8/2007
aspnet_PersonalizationAllUsers	dbo	6/8/2007
aspnet_PersonalizationPerUser	dbo	6/8/2007
aspnet_Profile	dbo	6/8/2007
aspnet_Roles	dbo	6/8/2007
aspnet_SchemaVersions	dbo	6/8/2007
aspnet_Users	dbo	6/8/2007
aspnet_UsersInRoles	dbo	6/8/2007
aspnet_WebEvent_Events	dbo	6/8/2007

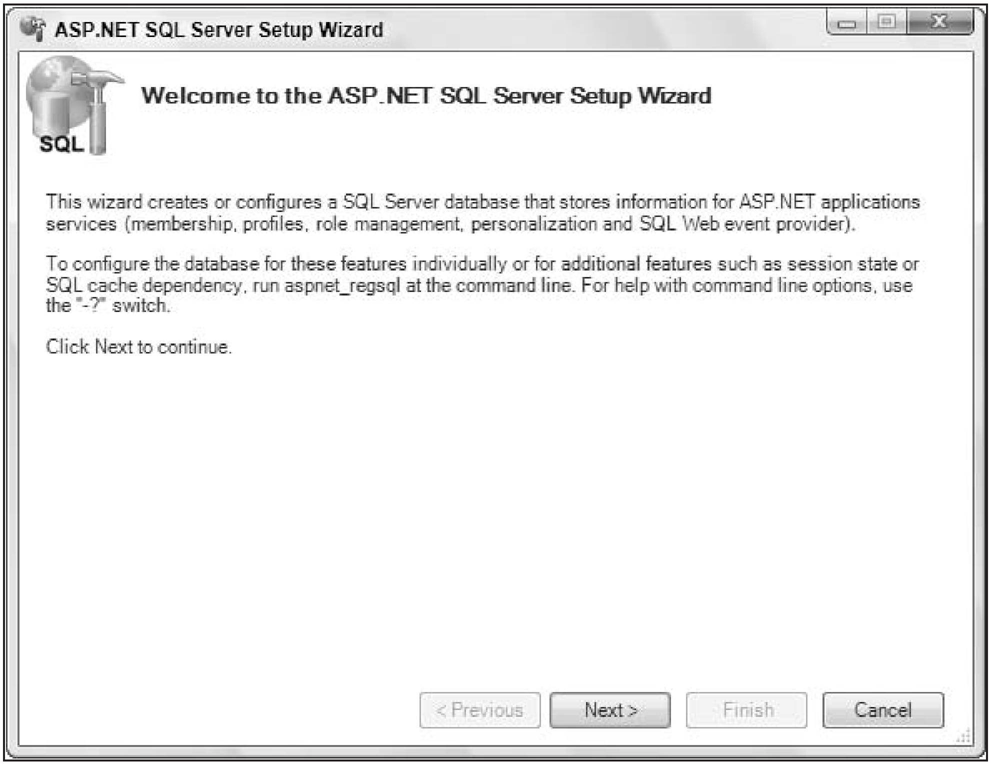
Figure 12-4

The ASP.NET SQL Server Setup Wizard GUI Tool

Instead of working with the tool through the command line, you can also work with a GUI version of the same wizard. To get at the GUI version, type the following at the Visual Studio command prompt:

```
aspnet_regsql.exe
```

At this point, the ASP.NET SQL Server Setup Wizard welcome screen appears, as shown in Figure 12-5.



Welcome to the ASP.NET SQL Server Setup Wizard

This wizard creates or configures a SQL Server database that stores information for ASP.NET applications services (membership, profiles, role management, personalization and SQL Web event provider).

To configure the database for these features individually or for additional features such as session state or SQL cache dependency, run aspnet_regsql at the command line. For help with command line options, use the "-?" switch.

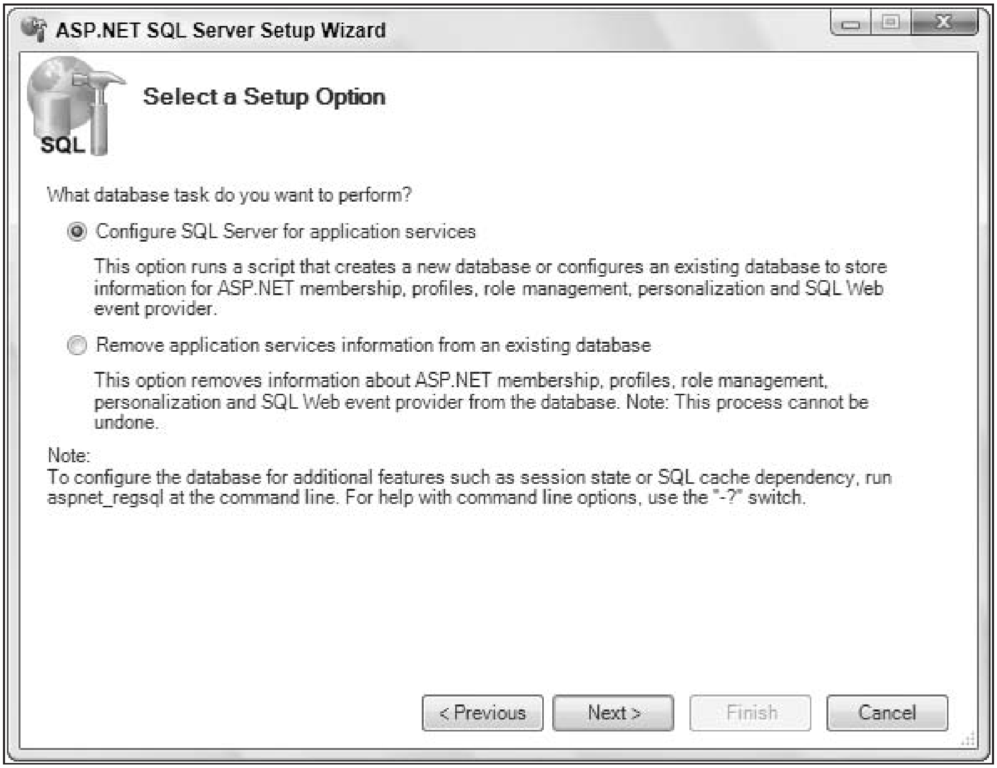
Click Next to continue.

Figure 12-5

Clicking Next gives you a new screen that offers two options: one to install management features into SQL Server and the other to remove them (see Figure 12-6).

From here, choose the Configure SQL Server for application services and click the Next button. The third screen (see Figure 12-7) asks for the login credentials to SQL Server and the name of the database to perform the operations. The Database option is `<default>` — meaning that the wizard creates a database called `aspnetdb`. If you want to choose a different folder, such as the application's database, choose the appropriate option.

After you have made your server and database selections, click Next. The screen shown in Figure 12-8 asks you to confirm your settings. If everything looks correct, click Next — otherwise, click Back and correct your settings.



What database task do you want to perform?

● Configure SQL Server for application services

This option runs a script that creates a new database or configures an existing database to store information for ASP.NET membership, profiles, role management, personalization and SQL Web event provider.

○ Remove application services information from an existing database

This option removes information about ASP.NET membership, profiles, role management, personalization and SQL Web event provider from the database. Note: This process cannot be undone.

Note:
To configure the database for additional features such as session state or SQL cache dependency, run aspnet_regsql at the command line. For help with command line options, use the "-?" switch.

Figure 12-6

When this is complete, you are notified that everything was set up correctly.

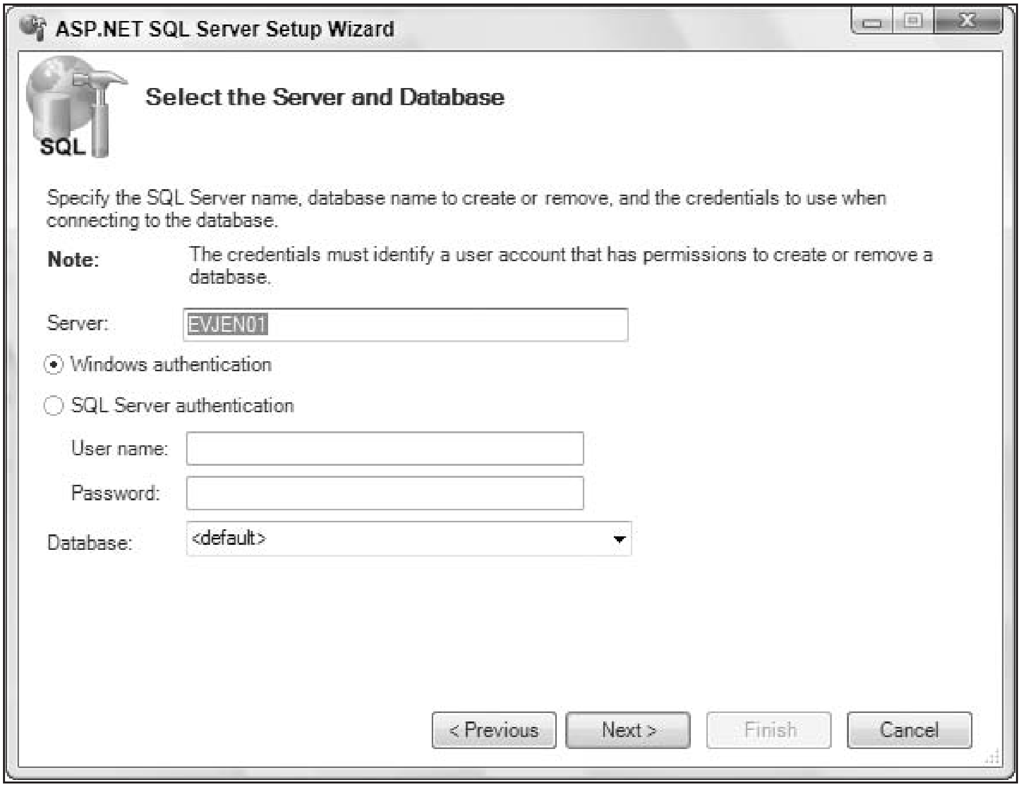
Connecting Your Default Provider to a New SQL Server Instance

After you set up the full-blown Microsoft SQL Server to work with the various systems provided by ASP.NET 3.5, you create a connection string to the database in your `machine.config` or `web.config` file. This is illustrated in Listing 12-1.

Listing 12-1: Changing the connection string in the machine.config.comments or your web.config file to work with SQL Server 2005

```
<configuration>

     <connectionStrings>
        <add name="LocalSqlServer"
         connectionString="Data Source=127.0.0.1;Integrated Security=SSPI" />
     </connectionStrings>

</configuration>
```

You may want to change the values provided if you are working with a remote instance of SQL Server rather than an instance that resides on the same server as the application. Changing this value in the



ASP.NET SQL Server Setup Wizard

Select the Server and Database

Specify the SQL Server name, database name to create or remove, and the credentials to use when connecting to the database.

Note: The credentials must identify a user account that has permissions to create or remove a database.

Server: EVJEN01

● Windows authentication

○ SQL Server authentication

User name:

Password:

Database: <default>

Figure 12-7

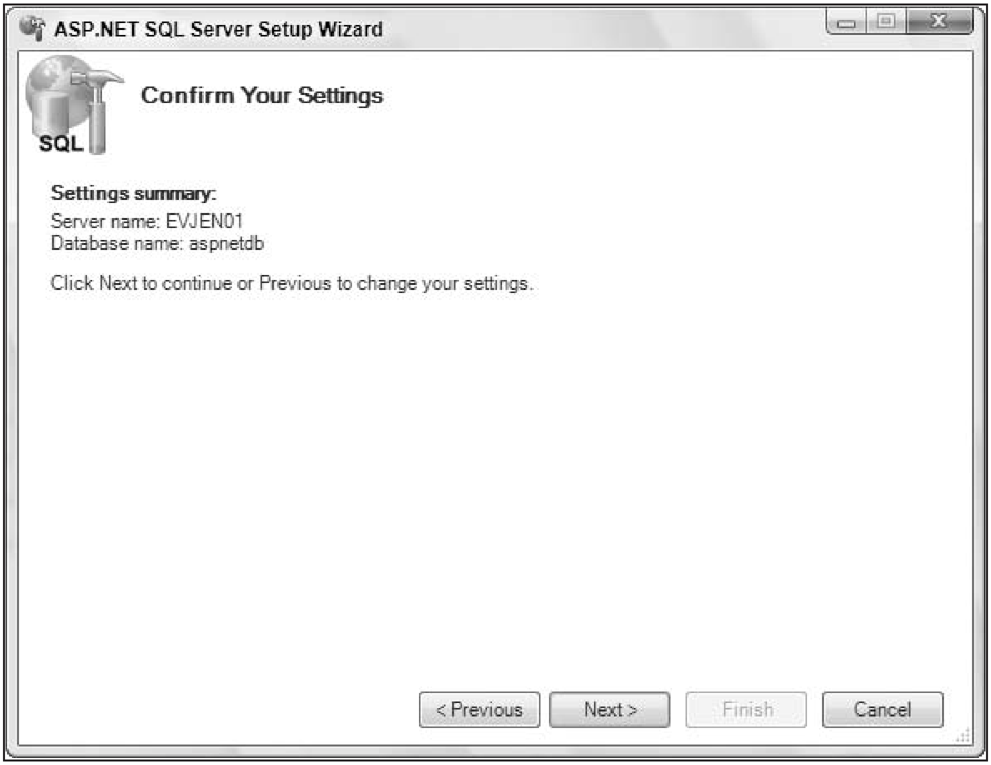
`machine.config` file changes how each and every ASP.NET application uses this provider. Applying this setting in the `web.config` file causes only the local application to work with this instance.

After the connection string is set up, look further in the `<providers>` section of the section you are going to work with. For instance, if you are using the membership provider, you want to work with the `<membership>` element in the configuration file. The settings to change the SQL Server are shown in Listing 12-2.

Listing 12-2: Altering the SQL Server used via configuration

```
<configuration>

  <connectionStrings>
     <add name="LocalSql2005Server"
      connectionString="Data Source=127.0.0.1;Integrated Security=SSPI" />
  </connectionStrings>

  <system.web>

        <membership defaultProvider="AspNetSql2005MembershipProvider">
            <providers>
```

```
                <add name="AspNetSql2005MembershipProvider"
                    type="System.Web.Security.SqlMembershipProvider,
                       System.Web, Version=2.0.0.0, Culture=neutral,
                       PublicKeyToken=b03f5f7f11d50a3a"
                    connectionStringName="LocalSql2005Server"
                    enablePasswordRetrieval="false"
                    enablePasswordReset="true"
                    requiresQuestionAndAnswer="true"
                    applicationName="/"
                    requiresUniqueEmail="false"
                    passwordFormat="Hashed"
                    maxInvalidPasswordAttempts="5"
                    minRequiredPasswordLength="7"
                    minRequiredNonalphanumericCharacters="1"
                    passwordAttemptWindow="10"
                    passwordStrengthRegularExpression="" />
            </providers>
        </membership>

  </system.web>

</configuration>
```


ASP.NET SQL Server Setup Wizard
SQL
< Previous
Next >
Finish
Cancel

Confirm Your Settings

Settings summary:

Server name: EVJEN01
Database name: aspnetdb

Click Next to continue or Previous to change your settings.

Figure 12-8

With these changes in place, the SQL Server 2005 instance is now one of the providers available for use with your applications. The name of this provider instance is `AspNetSql2005MembershipProvider`. You can see that this instance also uses the connection string of `LocalSql2005Server`, which was defined in Listing 12-1.

Pay attention to some important attribute declarations from Listing 12-2. The first is that the provider used by the membership system is defined via the `defaultProvider` attribute found in the main `<membership>` node. Using this attribute, you can specify whether the provider is one of the built-in providers or whether it is a custom provider that you have built yourself or received from a third party. With the code from Listing 12-2 in place, the membership provider now works with Microsoft SQL Server 2005 (as shown in this example) instead of the Microsoft SQL Server Express Edition files.

Next, you look at the providers that come built into the ASP.NET 3.5 install — starting with the membership system providers

Membership Providers

The membership system enables you to easily manage users in your ASP.NET applications. As with most of the systems provided in ASP.NET, it features a series of server controls that interact with a defined provider to either retrieve or record information to and from the data store defined by the provider. Because a provider exists between the server controls and the data stores where the data is retrieved and recorded, it is fairly trivial to have the controls work from an entirely different backend. You just change the underlying provider of the overall system (in this case, the membership system). This can be accomplished by a simple configuration change in the ASP.NET application. It really makes no difference to the server controls.

As previously stated, ASP.NET 3.5 provides two membership providers out of the box.

- `System.Web.Security.SqlMembershipProvider`: Provides you with the capability to use the membership system to connect to Microsoft's SQL Server 2000/2005 as well as with Microsoft SQL Server Express Edition.
- `System.Web.Security.ActiveDirectoryMembershipProvider`: Provides you with the capability to use the membership system to connect to Microsoft's Active Directory.

Both of these membership provider classes inherit from the `MembershipProvider` base class, as illustrated in Figure 12-9.

Next, you review each of these providers.

System.Web.Security.SqlMembershipProvider

The default provider is the `SqlMembershipProvider` instance. You find this default declaration for every ASP.NET application that resides on the application server in the `machine.config` file. This file is found in `C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG`. Listing 12-3 shows the definition of this provider, which is located in the `machine.config` file.

Listing 12-3: A SqlMembershipProvider instance declaration

```
<configuration>
   <system.web>
```

```
        <membership>
            <providers>
               <add name="AspNetSqlMembershipProvider"
                type="System.Web.Security.SqlMembershipProvider,
                   System.Web, Version=2.0.0.0, Culture=neutral,
                   PublicKeyToken=b03f5f7f11d50a3a"
                connectionStringName="LocalSqlServer"
                enablePasswordRetrieval="false" enablePasswordReset="true"
                requiresQuestionAndAnswer="true" applicationName="/"
                requiresUniqueEmail="false" passwordFormat="Hashed"
                maxInvalidPasswordAttempts="5" minRequiredPasswordLength="7"
                minRequiredNonalphanumericCharacters="1" passwordAttemptWindow="10"
                passwordStrengthRegularExpression=""/>
            </providers>
        </membership>
    </system.web>
</configuration>
```

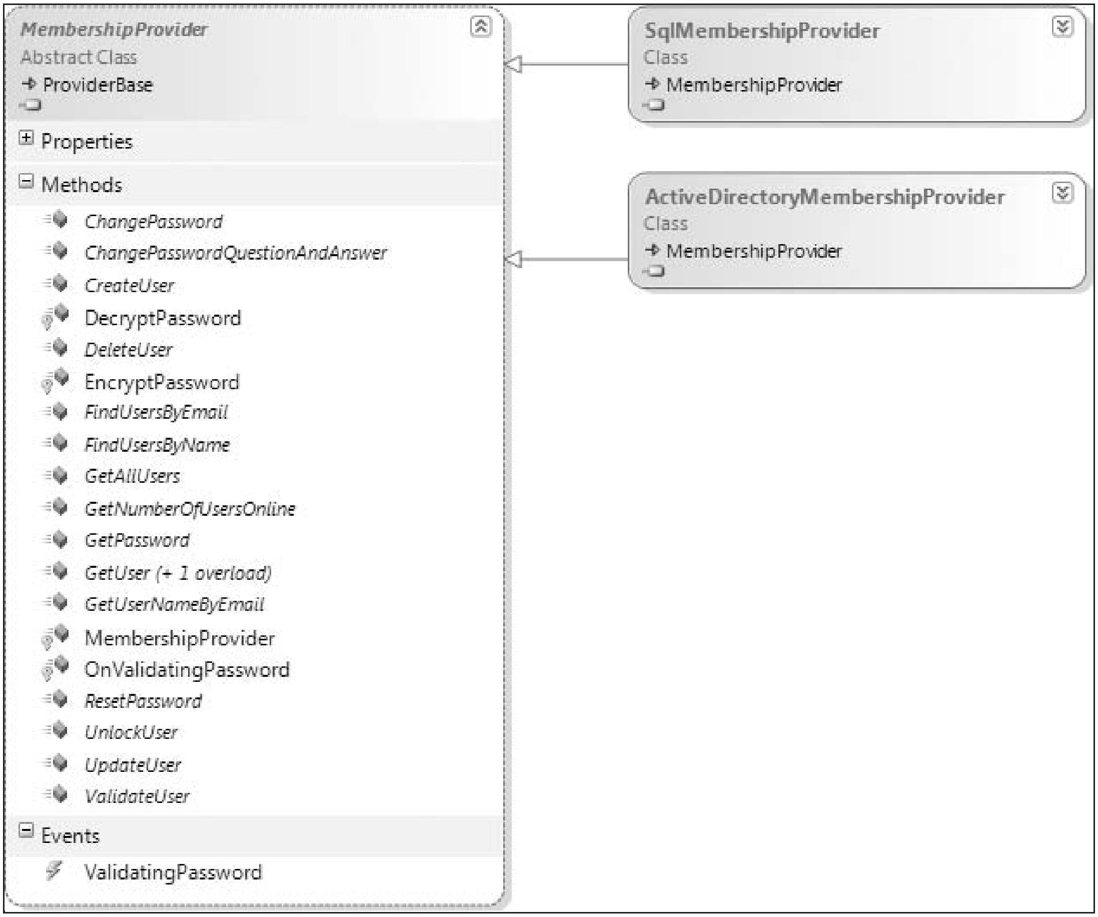

MembershipProvider
Abstract Class
ProviderBase
Properties
Methods
ChangePassword
ChangePasswordQuestionAndAnswer
CreateUser
DecryptPassword
DeleteUser
EncryptPassword
FindUsersByEmail
FindUsersByName
GetAllUsers
GetNumberOfUsersOnline
GetPassword
GetUser (+ 1 overload)
GetUserNameByEmail
MembershipProvider
OnValidatingPassword
ResetPassword
UnlockUser
UpdateUser
ValidateUser
Events
ValidatingPassword
SqlMembershipProvider
Class
MembershipProvider
ActiveDirectoryMembershipProvider
Class
MembershipProvider

Figure 12-9

From this listing, you can see that a single instance of the `SqlMembershipProvider` object is defined in the `machine.config` file. This single instance is named `AspNetSqlMembershipProvider`. This is also where you find the default behavior settings for your membership system. By default, this provider is also configured to work with a SQL Server Express Edition instance rather than a full-blown version of SQL Server such as SQL Server 2000, 2005, or 2008. You can see this by looking at the defined `connectionStringName` property in the provider declaration from Listing 12-3. In this case, it is set to `LocalSqlServer`. `LocalSqlServer` is also defined in the `machine.config` file as shown in Listing 12-4.

Listing 12-4: The LocalSqlServer defined instance

```
<configuration>
  <connectionStrings>
    <clear />
    <add name="LocalSqlServer"
     connectionString="Data Source=.\SQLEXPRESS;Integrated Security=SSPI;
        AttachDBFilename=|DataDirectory|aspnetdb.mdf;User Instance=true"
     providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

You can see this connection string information is set for a local SQL Server Express Edition file (an `.mdf` file). Of course, you are not required to work with only these file types for the `SqlMembershipProvider` capabilities. Instead, you can also set it up to work with either Microsoft's SQL Server 7.0, 2000, 2005, or 2008 (as was previously shown).

System.Web.Security.ActiveDirectoryMembershipProvider

It is also possible for the membership system provided from ASP.NET 3.5 to connect this system to a Microsoft Active Directory instance or even Active Directory Application Mode (ADAM), which is a stand-alone directory product. Because the default membership provider is defined in the `machine.config` files at the `SqlMembershipProvider`, you must override these settings in your application's `web.config` file.

Before creating a defined instance of the `ActiveDirectoryMembershipProvider` in your `web.config` file, you have to define the connection string to the Active Directory store. This is illustrated in Listing 12-5.

Listing 12-5: Defining the connection string to the Active Directory store

```
<configuration>
   <connectionStrings>
      <add name="ADConnectionString"
       connectionString=
       "LDAP://domain.myAdServer.com/CN=Users,DC=domain,DC=testing,DC=com" />
   </connectionStrings>
</configuration>
```

With the connection in place, you can create an instance of the `ActiveDirecotryMembershipProvider` in your `web.config` file that associates itself to this connection string. This is illustrated in Listing 12-6.

Listing 12-6: Defining the ActiveDirectoryMembershipProvider instance

```
<configuration>

   <connectionStrings>
      <add name="ADConnectionString"
       connectionString=
       "LDAP://domain.myAdServer.com/CN=Users,DC=domain,DC=testing,DC=com" />
   </connectionStrings>

   <system.web>
     <membership
      defaultProvider="AspNetActiveDirectoryMembershipProvider">
      <providers>
        <add name="AspNetActiveDirectoryMembershipProvider"
          type="System.Web.Security.ActiveDirectoryMembershipProvider,
            System.Web, Version=2.0.0.0, Culture=neutral,
            PublicKeyToken=b03f5f7f11d50a3a"
          connectionStringName="ADConnectionString"
          connectionUserName="UserWithAppropriateRights"
          connectionPassword="PasswordForUser"
          connectionProtection="Secure"
          enablePasswordReset="true"
          enableSearchMethods="true"
          requiresQuestionAndAnswer="true"
          applicationName="/"
          description="Default AD connection"
          requiresUniqueEmail="false"
          clientSearchTimeout="30"
          serverSearchTimeout="30"
          attributeMapPasswordQuestion="department"
          attributeMapPasswordAnswer="division"
          attributeMapFailedPasswordAnswerCount="singleIntAttribute"
          attributeMapFailedPasswordAnswerTime="singleLargeIntAttribute"
          attributeMapFailedPassswordAnswerLockoutTime="singleLargeIntAttribute"
          maxInvalidPasswordAttemps = "5"
          passwordAttemptWindow = "10"
          passwordAnswerAttemptLockoutDuration = "30"
          minRequiredPasswordLength="7"
          minRequiredNonalphanumericCharacters="1"
          passwordStrengthRegularExpression="
            @\"(?=.{6,})(?=(.*\d){1,})(?=(.*\W){1,})" />
        />
      </providers>
     </membership>
   </system.web>

</configuration>
```

Although not all these attributes are required, this list provides you with the available attributes of the `ActiveDirectoryMembershipProvider`. In fact, you can easily declare the instance in its simplest form, as shown here:

```
<membership defaultProvider="AspNetActiveDirectoryMembershipProvider">
  <providers>
    <add name="AspNetActiveDirectoryMembershipProvider"
      type="System.Web.Security.ActiveDirectoryMembershipProvider,
        System.Web, Version=2.0.0.0, Culture=neutral,
        PublicKeyToken=b03f5f7f11d50a3a"
      connectionStringName="ADConnectionString" />
  </providers>
</membership>
```

Again, with either the `SqlMembershipProvider` or the `ActiveDirectoryMembershipProvider` in place, the membership system server controls (such as the Login server control) as well as the membership API, once configured, will record and retrieve their information via the provider you have established. That is the power of the provider model that the ASP.NET team has established. You continue to see this power as you learn about the rest of the providers detailed in this chapter.

Role Providers

After a user is logged into the system (possibly using the ASP.NET membership system), the ASP.NET role management system enables you to work with the role of that user to authorize him for a particular access to the overall application. The role management system in ASP.NET 3.5, as with the other systems, has a set of providers to store and retrieve role information in an easy manner. This, of course, doesn't mean that you are bound to one of the three available providers in the role management system. Instead, you can extend one of the established providers or even create your own custom provider.

By default, ASP.NET 3.5 offers three providers for the role management system. These providers are defined in the following list:

- ☐ `System.Web.Security.SqlRoleProvider`: Provides you with the capability to use the ASP.NET role management system to connect to Microsoft's SQL Server 2000/2005/2008 as well as to Microsoft SQL Server Express Edition.
- ☐ `System.Web.Security.WindowsTokenRoleProvider`: Provides you with the capability to connect the ASP.NET role management system to the built-in Windows security group system.
- ☐ `System.Web.Security.AuthorizationStoreRoleProvider`: Provides you with the capability to connect the ASP.NET role management system to either an XML file, Active Directory, or in an Active Directory Application Mode (ADAM) store.

These three classes for role management inherit from the `RoleProvider` base class. This is illustrated in Figure 12-10.

System.Web.Security.SqlRoleProvider

The role management system in ASP.NET uses SQL Server Express Edition files by default (just as the membership system does). The connection to the SQL Server Express file uses `SqlRoleProvider`, but you can just as easily configure your SQL Server 7.0, 2000, 2005, or 2008 server to work with the role

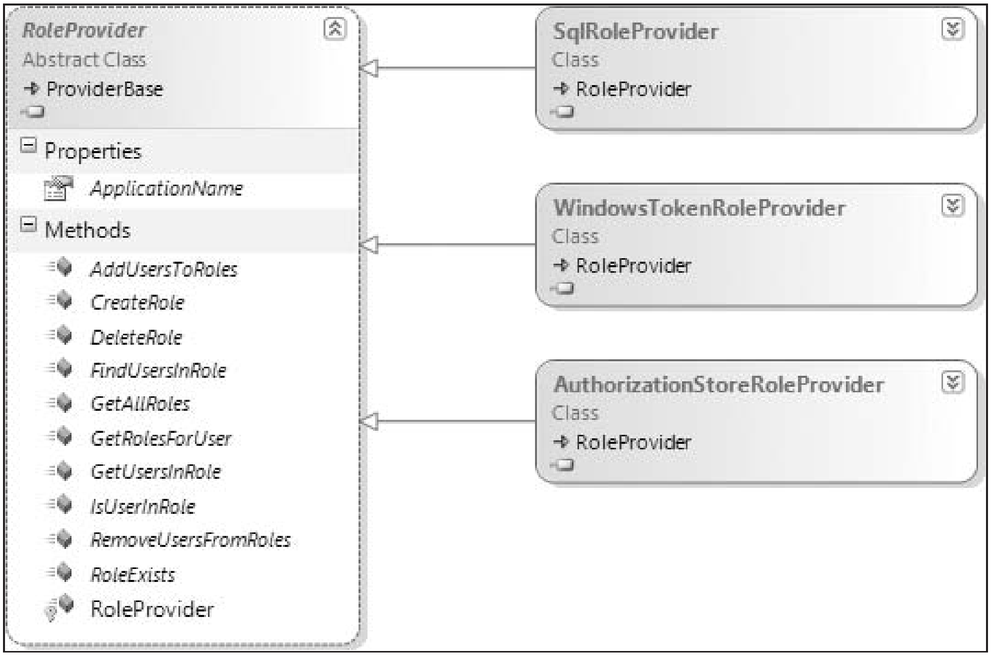

RoleProvider
Abstract Class
ProviderBase
Properties
ApplicationName
Methods
AddUsersToRoles
CreateRole
DeleteRole
FindUsersInRole
GetAllRoles
GetRolesForUser
GetUsersInRole
IsUserInRole
RemoveUsersFromRoles
RoleExists
RoleProvider
SqlRoleProvider
Class
RoleProvider
WindowsTokenRoleProvider
Class
RoleProvider
AuthorizationStoreRoleProvider
Class
RoleProvider

Figure 12-10

management system through `SqlRoleProvider`. The procedure for setting up your full-blown SQL Server is described in the beginning of this chapter.

Looking at the `SqlRoleProvider` instance in the `machine.config.comments` file, you will notice the syntax as defined in Listing 12-7. The `machine.config.comments` file provides documentation on the `machine.config` as well as showing you the details of the default settings that are baked into the ASP .NET Framework.

Listing 12-7: A SqlRoleProvider instance declaration

```
<configuration>
  <system.web>
     <roleManager enabled="false" cacheRolesInCookie="false"
      cookieName=".ASPXROLES" cookieTimeout="30" cookiePath="/"
      cookieRequireSSL="false" cookieSlidingExpiration="true"
      cookieProtection="All" defaultProvider="AspNetSqlRoleProvider"
      createPersistentCookie="false" maxCachedResults="25">
        <providers>
           <add name="AspNetSqlRoleProvider"
            connectionStringName="LocalSqlServer" applicationName="/"
            type="System.Web.Security.SqlRoleProvider,
            System.Web, Version=2.0.0.0, Culture=neutral,
            PublicKeyToken=b03f5f7f11d50a3a" />
        </providers>
     </roleManager>
   </system.web>
</configuration>
```

As stated, this is part of the default `<roleManager>` declaration that is baked into the overall ASP.NET Framework (note again that you can change any of these defaults by making a new declaration in your `web.config` file). As you can see, role management is disabled by default through the `enabled` attribute found in the `<roleManager>` node (it is set to `false` by default). Also, pay attention to the `defaultProvider` attribute in the `<roleManager>` element. In this case, it is set to `AspNetSqlRoleProvider`. This provider is defined in the same code example. To connect to the Microsoft SQL Server 2005 instance that was defined earlier (in the membership system examples), you can use the syntax shown in Listing 12-8.

Listing 12-8: Connecting the role management system to SQL Server 2005

```
<configuration>

  <connectionStrings>
     <add name="LocalSql2005Server"
      connectionString="Data Source=127.0.0.1;Integrated Security=SSPI" />
  </connectionStrings>

  <system.web>
        <roleManager enabled="true" cacheRolesInCookie="true"
         cookieName=".ASPXROLES" cookieTimeout="30" cookiePath="/"
         cookieRequireSSL="false" cookieSlidingExpiration="true"
         cookieProtection="All" defaultProvider="AspNetSqlRoleProvider"
         createPersistentCookie="false" maxCachedResults="25">
            <providers>
                <clear />
                <add connectionStringName="LocalSql2005Server" applicationName="/"
                 name="AspNetSqlRoleProvider"
                 type="System.Web.Security.SqlRoleProvider, System.Web,
                   Version=2.0.0.0, Culture=neutral,
                   PublicKeyToken=b03f5f7f11d50a3a" />
            </providers>
        </roleManager>
   </system.web>

</configuration>
```

With this in place, you can now connect to SQL Server 2005. Next is a review of the second provider available to the role management system.

System.Web.Security.WindowsTokenRoleProvider

The Windows operating system has a role system built into it. This Windows security group system is an ideal system to use when you are working with intranet-based applications where you might have all users already in defined roles. This, of course, works best if you have anonymous authentication turned off for your ASP.NET application, and you have configured your application to use Windows Authentication.

Windows Authentication for ASP.NET applications is discussed in Chapter 21.

Some limitations exist when using `WindowsTokenRoleProvider`. This is a read-only provider because ASP.NET is not allowed to modify the settings applied in the Windows security group system. This means that not all the methods provided via the `RoleProvider` abstract class are usable when working with this provider. From the `WindowsTokenRoleProvider` class, the only methods you have at your disposal are `IsUserInRole` and `GetUsersInRole`.

To configure your `WindowsTokenRoleProvider` instance, you use the syntax defined in Listing 12-9.

Listing 12-9: A WindowsTokenRoleProvider instance

```
<configuration>
   <system.web>

      <authentication mode="Windows" />

      <roleManager defaultProvider="WindowsProvider"
       enabled="true"
       cacheRolesInCookie="false">
         <providers>
            <add
             name="WindowsProvider"
             type="System.Web.Security.WindowsTokenRoleProvider" />
         </providers>
      </roleManager>

   </system.web>
</configuration>
```

Remember that you have to declare the default provider using the `defaultProvider` attribute in the `<roleManager>` element to change the assigned provider from the `SqlRoleProvider` association.

System.Web.Security.AuthorizationStoreRoleProvider

The final role provider you have available to you from a default install of ASP.NET is `AuthorizationStoreRoleProvider`. This role provider class allows you to store roles inside of an Authorization Manager policy store. These types of stores are also referred to as AzMan stores. As with `WindowsTokenRoleProvider`, `AuthorizationStoreRoleProvider` is a bit limited because it is unable to support any AzMan business rules.

To use `AuthorizationStoreRoleProvider`, you must first make a connection in your `web.config` file to the XML data store used by AzMan. This is illustrated in Listing 12-10.

Listing 12-10: Making a connection to the AzMan policy store

```
<configuration>
  <connectionStrings>
    <add name="LocalPolicyStore"
         connectionString="msxml://~\App_Data\datafilename.xml" />
  </connectionStrings>
</configuration>
```

Note that when working with these XML-based policy files, it is best to store them in the App_Data folder. Files stored in the App_Data folder cannot be pulled up in the browser.

After the connection string is in place, the next step is to configure your `AuthorizationStoreRoleProvider` instance. This takes the syntax defined in Listing 12-11.

Listing 12-11: Defining the AuthorizationStoreRoleProvider instance

```
<configuration>

  <connectionStrings>
    <add name="MyLocalPolicyStore"
         connectionString="msxml://~\App_Data\datafilename.xml" />
  </connectionStrings>

  <system.web>

    <authentication mode="Windows" />
    <identity impersonate="true" />

    <roleManager defaultProvider="AuthorizationStoreRoleProvider"
      enabled="true"
      cacheRolesInCookie="true"
      cookieName=".ASPROLES"
      cookieTimeout="30"
      cookiePath="/"
      cookieRequireSSL="false"
      cookieSlidingExpiration="true"
      cookieProtection="All" >
      <providers>
        <clear />
          <add
            name="AuthorizationStoreRoleProvider"
            type="System.Web.Security.AuthorizationStoreRoleProvider"
            connectionStringName="MyLocalPolicyStore"
            applicationName="SampleApplication"
            cacheRefreshInterval="60"
            scopeName="" />
      </providers>
    </roleManager>

  </system.web>

</configuration>
```

Next, this chapter reviews the single personalization provider available in ASP.NET 3.5.

The Personalization Provider

As with the membership system found in ASP.NET, the personalization system (also referred to as the profile system) is another system that is based on the provider model. This system makes associations between the end user viewing the application and any data points stored centrally that are specific to that user. As stated, these personalization properties are stored and maintained on a per-user basis. ASP.NET provides a single provider for data storage. This provider is detailed here:

- ❑ `System.Web.Profile.SqlProfileProvider`: Provides you with the capability to use the ASP.NET personalization system to connect to Microsoft's SQL Server 2000/2005/2008 as well as to the new Microsoft SQL Server Express Edition.

This single class for the personalization system inherits from the `ProfileProvider` base class. This is illustrated in Figure 12-11.

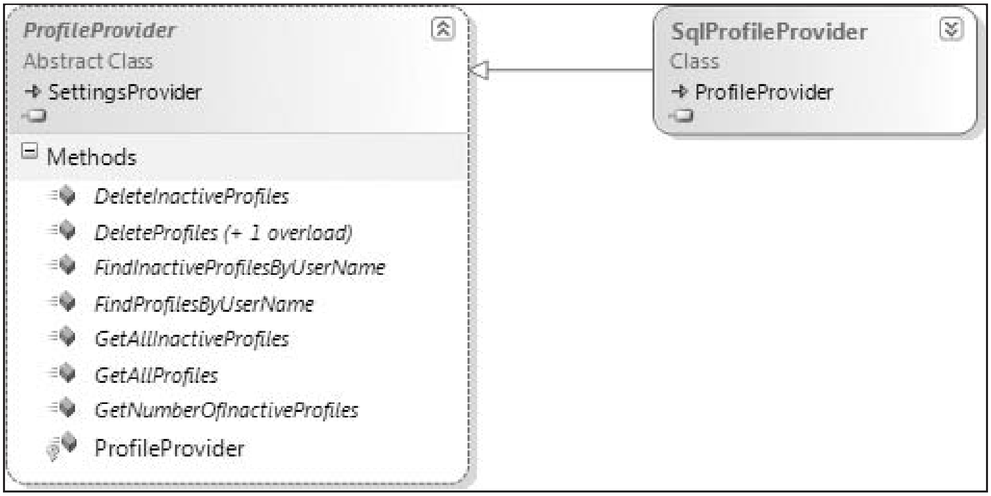

ProfileProvider
Abstract Class
→ SettingsProvider
Methods
DeleteInactiveProfiles
DeleteProfiles (+ 1 overload)
FindInactiveProfilesByUserName
FindProfilesByUserName
GetAllInactiveProfiles
GetAllProfiles
GetNumberOfInactiveProfiles
ProfileProvider
SqlProfileProvider
Class
→ ProfileProvider

Figure 12-11

As with the other providers covered so far, `SqlProfileProvider` connects to a Microsoft SQL Server Express Edition file by default. Although this is the default, you can change the connection to work with either SQL Server 7.0, 2000, 2005, or 2008. For instance, if you are connecting to a SQL Server 2005 database, you define your connection in the `web.config` and then associate your `SqlProfileProvider` declaration to this connection string. This scenario is presented in Listing 12-12.

Listing 12-12: Connecting the SqlProfileProvider to SQL Server 2005

```
<configuration>

     <connectionStrings>
        <add name="LocalSql2005Server"
         connectionString="Data Source=127.0.0.1;Integrated Security=SSPI" />
     </connectionStrings>

     <system.web>

     <profile>
        <providers>
           <clear />
           <add name="AspNetSql2005ProfileProvider"
            connectionStringName="LocalSql2005Server" applicationName="/"
            type="System.Web.Profile.SqlProfileProvider, System.Web,
              Version=2.0.0.0, Culture=neutral,
              PublicKeyToken=b03f5f7f11d50a3a" />
        </providers>
```

Continued

```
            <properties>
              <add name="FirstName" />
              <add name="LastName" />
              <add name="LastVisited" />
              <add name="Age" />
              <add name="Member" />
            </properties>
        </profile>

       </system.web>

   </configuration>
```

Remember that to store profile information in your SQL Server database, you have to configure this database so the proper tables, stored procedures, and other items are created. This task was discussed earlier in the chapter.

The SiteMap Provider

Similar to the personalization provider just discussed, ASP.NET 3.5 provides a single provider to work with sitemaps. Sitemaps are what ASP.NET uses to provide you with a centralized way of maintaining site navigation. By default, the definition of a Web application's navigation is located in a structured XML file. The sitemap provider lets you interact with this XML file, the `.sitemap` file, which you create for your application. The provider available for sitemaps is `System.Web.XmlSiteMapProvider`, which provides you with the capability to use the ASP.NET navigation system to connect to an XML-based file.

This single class for the sitemap system inherits from the `StaticSiteMapProvider` base class, which is a partial implementation of the `SiteMapProvider` base class. This is illustrated in Figure 12-12.

This is the first provider introduced so far that does not connect to a SQL Server database by default. Instead, this provider is designed to work with a static XML file. This XML file uses a particular schema and is covered in considerable detail in Chapter 14.

The code required to configure `XmlSiteMapProvider` is presented in Listing 12-13.

Listing 12-13: Defining an XmlSiteMapProvider instance in the web.config

```
   <configuration>
      <system.web>

         <siteMap defaultProvider="MyXmlSiteMapProvider" enabled="true">
            <providers>
               <add name="MyXmlSiteMapProvider"
                description="SiteMap provider that reads in .sitemap files."
                type="System.Web.XmlSiteMapProvider, System.Web, Version=2.0.0.0,
                  Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
                siteMapFile="AnotherWeb.sitemap" />
            </providers>
         </siteMap>

      </system.web>
   </configuration>
```

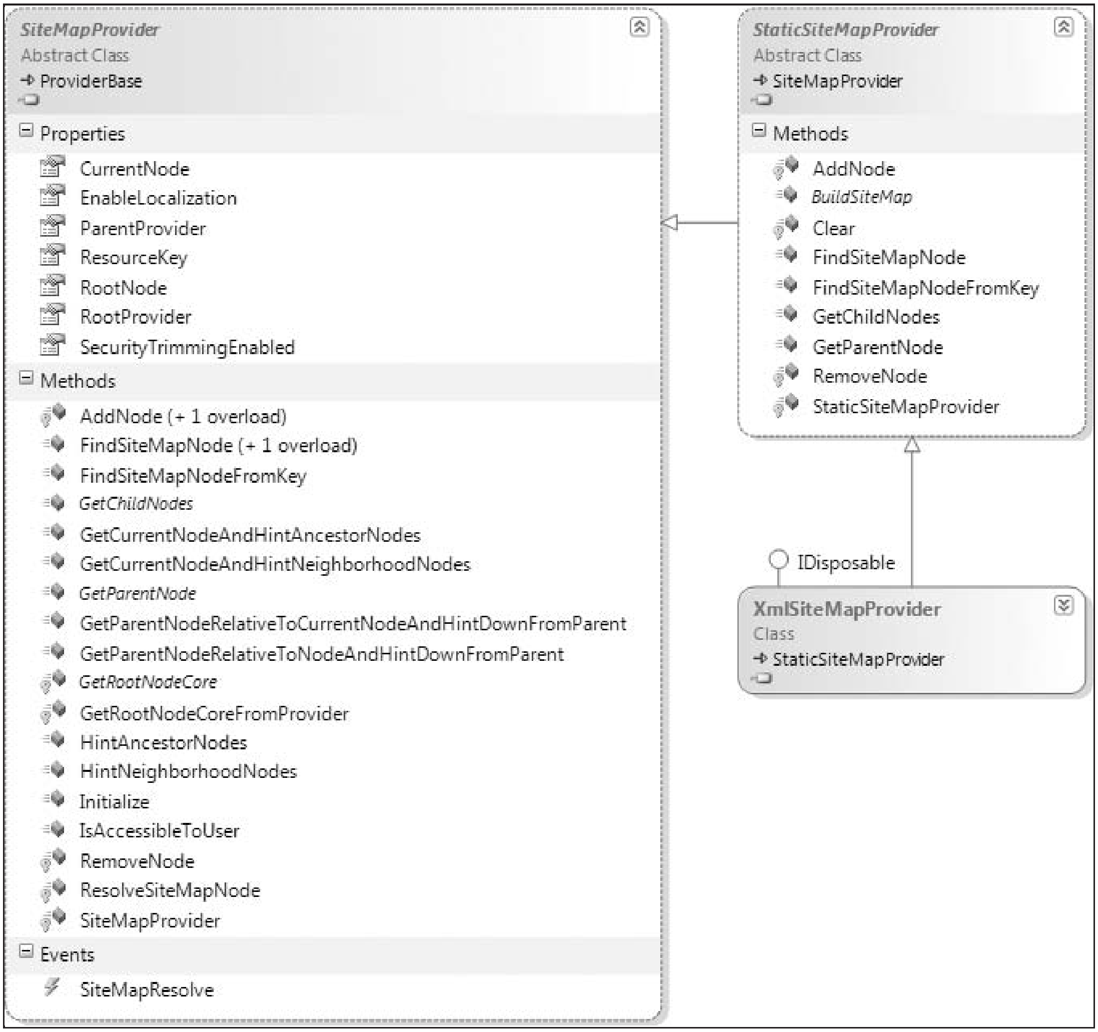

SiteMapProvider
Abstract Class
ProviderBase
Properties
CurrentNode
EnableLocalization
ParentProvider
ResourceKey
RootNode
RootProvider
SecurityTrimmingEnabled
Methods
AddNode (+ 1 overload)
FindSiteMapNode (+ 1 overload)
FindSiteMapNodeFromKey
GetChildNodes
GetCurrentNodeAndHintAncestorNodes
GetCurrentNodeAndHintNeighborhoodNodes
GetParentNode
GetParentNodeRelativeToCurrentNodeAndHintDownFromParent
GetParentNodeRelativeToNodeAndHintDownFromParent
GetRootNodeCore
GetRootNodeCoreFromProvider
HintAncestorNodes
HintNeighborhoodNodes
Initialize
IsAccessibleToUser
RemoveNode
ResolveSiteMapNode
SiteMapProvider
Events
SiteMapResolve
StaticSiteMapProvider
Abstract Class
SiteMapProvider
Methods
AddNode
BuildSiteMap
Clear
FindSiteMapNode
FindSiteMapNodeFromKey
GetChildNodes
GetParentNode
RemoveNode
StaticSiteMapProvider
IDisposable
XmlSiteMapProvider
Class
StaticSiteMapProvider

Figure 12-12

The `XmlSiteMapProvider` allows only a single root element in the strictly designed `web.sitemap` file. The default file name of the XML file it is looking for is `web.sitemap`, although you can change this default setting (as you can see in Listing 12-13) by using the `siteMapFile` attribute within the provider declaration in the `web.config` file.

SessionState Providers

As mentioned in the beginning of the chapter, an original concept of a provider model existed when the idea of managing session state in different ways was first introduced with ASP.NET 1.*x*. The available modes of storing session state for your users include `InProc`, `StateServer`, `SQLServer`, or even `Custom`.

Each mode has definite pros and cons associated with it, and you should examine each option thoroughly when deciding which session state mode to use.

You can find more information on these session state modes in Chapter 22.

This provider model is a bit different from the others discussed so far in this chapter. The `SessionStateModule` class is a handler provided to load one of the available session state modes. Each of these modes is defined here:

- `System.Web.SessionState.InProcSessionStateStore`: Provides you with the capability to store sessions in the same process as the ASP.NET worker process. This is by far the best-performing method of session state management.
- `System.Web.SessionState.OutOfProcSessionStateStore`: Provides you with the capability to store sessions in a process separate from the ASP.NET worker process. This mode is a little more secure, but a little worse in performance than the InProc mode.
- `System.Web.SessionState.SqlSessionStateStore`: Provides you with the capability to store sessions in SQL Server. This is by far the most secure method of storing sessions, but it is the worst performing mode of the three available methods.

These three modes for session state management are illustrated in Figure 12-13.

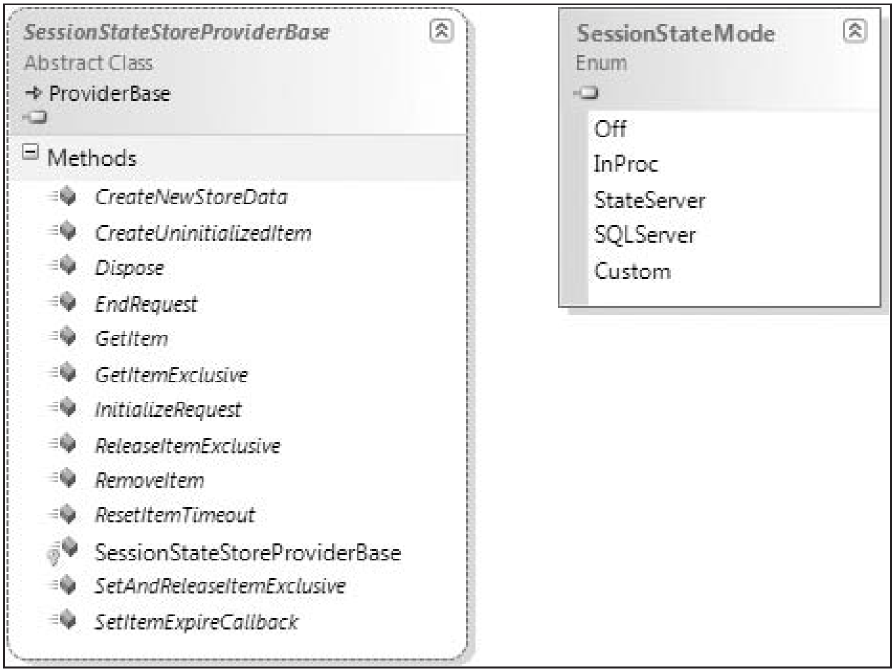

SessionStateStoreProviderBase
Abstract Class
ProviderBase
Methods
CreateNewStoreData
CreateUninitializedItem
Dispose
EndRequest
GetItem
GetItemExclusive
InitializeRequest
ReleaseItemExclusive
RemoveItem
ResetItemTimeout
SessionStateStoreProviderBase
SetAndReleaseItemExclusive
SetItemExpireCallback
SessionStateMode
Enum
Off
InProc
StateServer
SQLServer
Custom

Figure 12-13

Next, this chapter reviews each of the three modes that you can use out of the box in your ASP.NET 3.5 applications.

System.Web.SessionState.InProcSessionStateStore

The `InProcSessionStateStore` mode is the default mode for ASP.NET 1.*x* as well as for ASP.NET 2.0 and 3.5. In this mode, the sessions generated are held in the same process as that being used by the ASP.NET worker process (`aspnet_wp.exe` or `w3wp.exe`). This mode is the best performing, but some problems exist with this mode as well. Because the sessions are stored in the same process, whenever the worker process is recycled, all the sessions are destroyed. Worker processes can be recycled for many reasons (such as a change to the `web.config` file, the `Global.asax` file, or a setting in IIS that requires the process to be recycled after a set time period).

An example of the configuration in the `web.config` file for working in the InProc mode is presented in Listing 12-14.

Listing 12-14: Defining the InProc mode for session state management in the web.config

```
<configuration>
   <system.web>

      <sessionState mode="InProc">
      </sessionState>

   </system.web>
</configuration>
```

As you can see, this mode is rather simple. The next method reviewed is the out-of-process mode — also referred to as the StateServer mode.

System.Web.SessionState.OutOfProcSessionStateStore

In addition to the InProc mode, the StateServer mode is an out-of-process method of storing session state. This method does not perform as well as one that stores the sessions in the same process as the ASP.NET worker process. This makes sense because the method must jump process boundaries to work with the sessions you are employing. Although the performance is poorer than in the InProc mode, the `OutOfProcSessionStateStore` method is more reliable than running the sessions using `InProcSessionStateStore`. If your application's worker process recycles, the sessions that this application is working with are still maintained. This is a vital capability for those applications that are critically dependent upon sessions.

An example of using `OutOfProcSessionStateStore` is detailed in Listing 12-15.

Listing 12-15: Running sessions out of process using OutOfProcSessionStateStore

```
<configuration>
   <system.web>

      <sessionState mode="StateServer"
       stateConnectionString="tcpip=127.0.0.1:42424">
      </sessionState>

   </system.web>
</configuration>
```

When using the StateServer mode, you also must define where the sessions are stored using the `stateConnectionString` attribute. In this case, the local server is used, meaning that the sessions are stored on the same machine, but in an entirely separate process. You could have just as easily stored the sessions on a different server by providing the appropriate IP address as a value for this attribute. In addition to the IP address, note that port 42424 is used. This port is required when using the StateServer mode for sessions. Changing the port for the StateServer is detailed in Chapter 22.

System.Web.SessionState.SqlSessionStateStore

The final provider for session state management available to you in ASP.NET is the `SqlSessionStateStore`. This method is definitely the most resilient of the three available modes. With that said, however, it is also the worst performing of the three modes. It is important that you set up your database appropriately if you use this method of session state storage. Again, Chapter 22 shows you how to set up your database.

To configure your application to work with `SqlSessionStateStore`, you must configure the `web.config` as detailed in Listing 12-16.

Listing 12-16: Defining SqlSessionStateStore in the web.config file

```
<configuration>
   <system.web>

      <sessionState mode="SQLServer"
       allowCustomSqlDatabase="true"
       sqlConnectionString="Data Source=127.0.0.1;
         database=MyCustomASPStateDatabase;Integrated Security=SSPI">
      </sessionState>

   </system.web>
</configuration>
```

Next, you review the providers available for the Web events architecture.

Web Event Providers

Among all the available systems provided in ASP.NET 3.5, more providers are available for the health monitoring system than for any other system. The new health monitoring system enables ASP.NET application administrators to evaluate the health of a running ASP.NET application and to capture events (errors and other possible triggers) that can then be stored via one of the available providers. These events are referred to as *Web events*. A large list of events can be monitored via the health monitoring system, and this means that you can start recording items such as authentication failures/successes, all errors generated, ASP.NET worker process information, request data, response data, and more. Recording items means using one of the providers available to record to a data store of some kind.

Health monitoring in ASP.NET 3.5 is covered in Chapter 33.

By default, ASP.NET 3.5 offers seven possible providers for the health monitoring system. This is more than for any of the other ASP.NET systems. These providers are defined in the following list:

- ☐ `System.Web.Management.EventLogWebEventProvider`: Provides you with the capability to use the ASP.NET health monitoring system to record security operation errors and all other errors into the Windows event log.
- ☐ `System.Web.Management.SimpleMailWebEventProvider`: Provides you with the capability to use the ASP.NET health monitoring system to send error information in an e-mail.
- ☐ `System.Web.Management.TemplatedMailWebEventProvider`: Similar to the `SimpleMailWebEventProvider`, the `TemplatedMailWebEventProvider` class provides you with the capability to send error information in a templated e-mail. Templates are defined using a standard `.aspx` page.
- ☐ `System.Web.Management.SqlWebEventProvider`: Provides you with the capability to use the ASP.NET health monitoring system to store error information in SQL Server. As with the other SQL providers for the other systems in ASP.NET, the `SqlWebEventProvider` stores error information in SQL Server Express Edition by default.
- ☐ `System.Web.Management.TraceWebEventProvider`: Provides you with the capability to use the ASP.NET health monitoring system to send error information to the ASP.NET page tracing system.
- ☐ `System.Web.Management.IisTraceWebEventProvider`: Provides you with the capability to use the ASP.NET health monitoring system to send error information to the IIS tracing system.
- ☐ `System.Web.Management.WmiWebEventProvider`: Provides you with the capability to connect the new ASP.NET health monitoring system, the Windows Management Instrumentation (WMI) event provider.

These seven providers for the ASP.NET health monitoring system inherit from either the `WebEventProvider` base class, or the `BufferedWebEventProvider` (which, in turn, inherits from `WebEventProvider`). This is illustrated in Figure 12-14.

What is the difference between the `WebEventProvider` class and the `BufferedWebEventProvider`? The big difference is that the `WebEventProvider` writes events as they happen, whereas the `BufferedWebEventProvider` holds Web events until a collection of them is made. The collection is then written to the database or sent in an e-mail in a batch. If you use the `SqlWebEventProvider` class, you actually want this batch processing to occur rather than having the provider make a connection to the database and write to it for each Web event that occurs.

Next, this chapter looks at each of the seven available providers for the health monitoring system.

System.Web.Management.EventLogWebEventProvider

Traditionally, administrators and developers are used to reviewing system and application errors in the built-in Windows event log. The items in the event log can be viewed via the Event Viewer. This

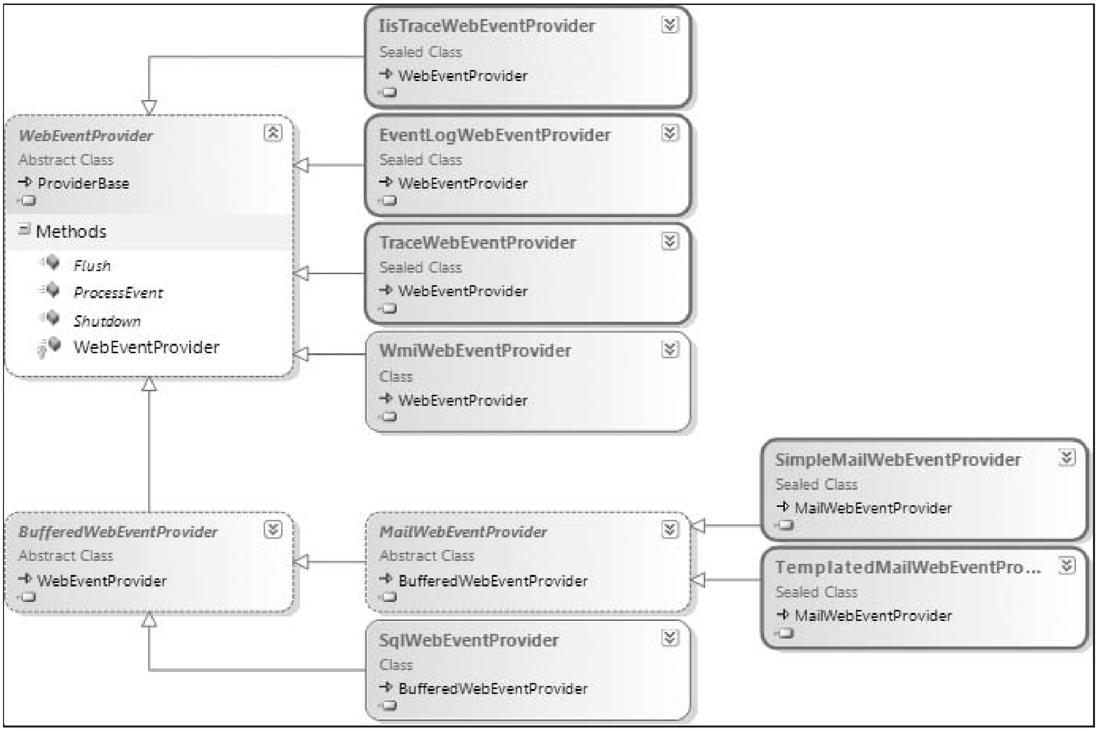

IisTraceWebEventProvider
Sealed Class
WebEventProvider
WebEventProvider
Abstract Class
ProviderBase
Methods
Flush
ProcessEvent
Shutdown
WebEventProvider
EventLogWebEventProvider
Sealed Class
WebEventProvider
TraceWebEventProvider
Sealed Class
WebEventProvider
WmiWebEventProvider
Class
WebEventProvider
SimpleMailWebEventProvider
Sealed Class
MailWebEventProvider
BufferedWebEventProvider
Abstract Class
WebEventProvider
MailWebEventProvider
Abstract Class
BufferedWebEventProvider
TemplatedMailWebEventPro...
Sealed Class
MailWebEventProvider
SqlWebEventProvider
Class
BufferedWebEventProvider

Figure 12-14

GUI-based tool for viewing events can be found by selecting Administration Tools in the Control Panel and then selecting Event Viewer.

By default, the health monitoring system uses the Windows event log to record the items that are already specified in the server's configuration files or items you have specified in the `web.config` file of your application. If you look in the `web.config.comments` file in the CONFIG folder of the Microsoft .NET Framework install on your server, you see that the `EventLogWebEventProvider` is detailed in this location. The code is presented in Listing 12-17.

Listing 12-17: The EventLogWebEventProvider declared in the web.config.comments file

```
<configuration>
   <system.web>

      <healthMonitoring heartbeatInterval="0" enabled="true">
         <bufferModes>
            <!-- Removed for clarity -->
         </bufferModes>
         <providers>
            <clear />
            <add name="EventLogProvider"
```

```
                type="System.Web.Management.EventLogWebEventProvider,
                  System.Web,Version=2.0.0.0,Culture=neutral,
                  PublicKeyToken=b03f5f7f11d50a3a" />
               <!-- Removed for clarity -->
            </providers>
            <profiles>
               <!-- Removed for clarity -->
            </profiles>
            <rules>
               <add name="All Errors Default" eventName="All Errors"
                provider="EventLogProvider" profile="Default" minInstances="1"
                maxLimit="Infinite" minInterval="00:01:00" custom="" />
               <add name="Failure Audits Default" eventName="Failure Audits"
                provider="EventLogProvider" profile="Default" minInstances="1"
                maxLimit="Infinite" minInterval="00:01:00" custom="" />
            </rules>
            <eventMappings>
               <!-- Removed for clarity -->
            </eventMappings>
         </healthMonitoring>

      </system.web>
   </configuration>
```

As you can see from Listing 12-17, a lot of possible settings can be applied in the health monitoring system. Depending on the rules and event mappings you have defined, these items are logged into the event log of the server that is hosting the application. Looking closely at the `<rules>` section of the listing, you can see that specific error types are assigned to be monitored. In this section, two types of errors are trapped in the health monitoring system — `All Errors Default` and `Failure Audits Default`.

When one of the errors defined in the `<rules>` section is triggered and captured by the health monitoring system, it is recorded. Where it is recorded depends upon the specified provider. The provider attribute used in the `<add>` element of the `<rules>` section determines this. In both cases in the example in Listing 12-17, you can see that the `EventLogProvider` is the assigned provider. This means that the Windows error log is used for recording the errors of both types.

> *As you work through the rest of the providers, note that the health monitoring system behaves differently when working with providers than the other systems that have been introduced in this chapter. Using the new health monitoring system in ASP.NET 3.5, you are able to assign more than one provider at a time. This means that you are able to specify in the* `web.config` *file that errors are logged not only into the Windows event log, but also into any other data store using any other provider you designate. Even for the same Web event type, you can assign the Web event to be recorded to the Windows event log and SQL Server at the same time (for example).*

System.Web.Management.SimpleMailWebEventProvider

Sometimes when errors occur in your applications, you as an administrator or a concerned developer want e-mail notification of the problem. In addition to recording events to disk using something such as the `EventLogWebEventProvider`, you can also have the error notification e-mailed to you using the `SimpleMailWebEventProvider`. As it states in the provider name, the e-mail is a simply constructed one. Listing 12-18 shows you how you would go about adding e-mail notification in addition to writing the errors to the Windows event log.

Listing 12-18: The SimpleMailWebEventProvider definition

```
<configuration>
   <system.web>

      <healthMonitoring heartbeatInterval="0" enabled="true">
         <bufferModes>
            <add name="Website Error Notification"
             maxBufferSize="100"
             maxFlushSize="20"
             urgentFlushThreshold="1"
             regularFlushInterval="Infinite"
             urgentFlushInterval="00:01:00"
             maxBufferThreads="1" />
         </bufferModes>
         <providers>
            <clear />
            <add name="EventLogProvider"
             type="System.Web.Management.EventLogWebEventProvider,
               System.Web,Version=2.0.0.0,Culture=neutral,
               PublicKeyToken=b03f5f7f11d50a3a" />
            <add name="SimpleMailProvider"
             type="System.Web.Management.SimpleMailWebEventProvider,
              System.Web, Version=2.0.0.0, Culture=neutral,
              PublicKeyToken=b03f5f7f11d50a3a"
             from="website@company.com"
             to="admin@company.com"
             cc="adminLevel2@company.com
             bcc="director@company.com"
             bodyHeader="Warning!"
             bodyFooter="Please investigate ASAP."
             subjectPrefix="Action required."
             buffer="true"
             bufferMode="Website Error Notification"
             maxEventLength="4096"
             maxMessagesPerNotification="1" />
         </providers>
         <profiles>
            <!-- Removed for clarity -->
         </profiles>
         <rules>
            <add name="All Errors Default" eventName="All Errors"
             provider="EventLogProvider" profile="Default" minInstances="1"
             maxLimit="Infinite" minInterval="00:01:00" custom="" />
            <add name="Failure Audits Default" eventName="Failure Audits"
             provider="EventLogProvider" profile="Default" minInstances="1"
             maxLimit="Infinite" minInterval="00:01:00" custom="" />
            <add name="All Errors Simple Mail" eventName="All Errors"
             provider="SimpleMailProvider" profile="Default" />
            <add name="Failure Audits Default" eventName="Failure Audits"
             provider="SimpleMailProvider" profile="Default" />
         </rules>
         <eventMappings>
```

```
            <!-- Removed for clarity -->
         </eventMappings>
      </healthMonitoring>

   </system.web>
</configuration>
```

In this example, the errors that occur are captured and not only written to the event log, but are also e-mailed to the end users specified in the provider definition. One very interesting point of the `SimpleMailWebEventProvider` is that this class inherits from the `BufferedWebEventProvider` instead of from the `WebEventProvider` as the `EventLogWebEventProvider` does. Inheriting from the `BufferedWebEventProvider` means that you can have the health monitoring system build a collection of error notifications before sending them on. The `<bufferModes>` section defines how the buffering works.

System.Web.Management.TemplatedMailWebEventProvider

The aforementioned `SimpleMailWebEventProvider` does exactly what its name states — it sends out a simple, text-based e-mail. To send out a more artistically crafted e-mail that contains even more information, you can use the `TemplatedMailWebEventProvider`. Just like the `SimpleMailWebEventProvider`, you simply define the provider appropriately in the `<healthMonitoring>` section. The model for this is presented in Listing 12-19.

Listing 12-19: The TemplatedMailWebEventProvider definition

```
<providers>
   <clear />
   <add name="EventLogProvider"
    type="System.Web.Management.EventLogWebEventProvider,
      System.Web,Version=2.0.0.0,Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a" />
   <add name="TemplatedMailProvider"
    type="System.Web.Management.TemplatedMailWebEventProvider,
      System.Web, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a"
     template="../mailtemplates/errornotification.aspx"
     from="website@company.com"
     to="admin@company.com"
     cc="adminLevel2@company.com
     bcc="director@company.com"
     bodyHeader="Warning!"
     bodyFooter="Please investigate ASAP."
     subjectPrefix="Action required."
     buffer="true"
     bufferMode="Website Error Notification"
     maxEventLength="4096"
     maxMessagesPerNotification="1" />
 </providers>
```

The big difference between this provider declaration and the `SimpleMailWebEventProvider` is shown in bold in Listing 12-19. The `TemplatedMailWebEventProvider` has a `template` attribute that specifies the location of the template to use for the e-mail that is created and sent from the health monitoring system.

Again, details on using the templated e-mail notification in the health monitoring system appear in Chapter 33.

System.Web.Management.SqlWebEventProvider

In many instances, you may want to write to disk when you are trapping and recording the Web events that occur in your application. The `EventLogWebEventProvider` is an excellent provider because it writes these Web events to the Windows event log on your behalf. However, in some instances, you may want to write these Web events to disk elsewhere. In this case, a good alternative is writing these Web events to SQL Server instead (or even in addition to the writing to an event log).

Writing to SQL Server gives you some benefits over writing to the Windows event log. When your application is running in a Web farm, you might want all the errors that occur across the farm to be written to a single location. In this case, it makes sense to write all Web events that are trapped via the health monitoring system to a SQL Server instance to which all the servers in the Web farm can connect.

By default, the `SqlWebEventProvider` (like the other SQL Server-based providers covered so far in this chapter) uses SQL Server Express Edition as its underlying database. To connect to the full-blown version of SQL Server instead, you need a defined connection as shown in Listing 12-20.

Listing 12-20: The LocalSql2005Server defined instance

```
<configuration>

     <connectionStrings>
        <add name="LocalSql2005Server"
         connectionString="Data Source=127.0.0.1;Integrated Security=SSPI" />
     </connectionStrings>

</configuration>
```

With this connection in place, the next step is to use this connection in your `SqlWebEventProvider` declaration in the `web.config` file. This is illustrated in Listing 12-21.

Listing 12-21: Writing Web events to SQL Server 2005 using the SqlWebEventProvider

```
<configuration>
   <system.web>

      <healthMonitoring>

         <!-- Other nodes removed for clarity -->

         <providers>
            <clear />
            <add name="SqlWebEventProvider"
             type="System.Web.Management.SqlWebEventProvider,System.Web"
             connectionStringName="LocalSql2005Server"
             maxEventDetailsLength="1073741823"
             buffer="true"
             bufferMode="SQL Analysis" />
         </providers>
      </healthMonitoring>

   </system.web>
</configuration>
```

Events are now recorded in SQL Server 2005 on your behalf. The nice thing about the `SqlWebEventProvider` is that, as with the `SimpleMailWebEventProvider` and the `TemplatedMailWebEventProvider`, the `SqlWebEventProvider` inherits from the `BufferedWebEventProvider`. This means that the Web events can be written in batches as opposed to one by one. This is done by using the `buffer` and `bufferMode` attributes in the provider declaration. It works in conjunction with the settings applied in the `<bufferModes>` section of the `<healthMonitoring>` declarations.

System.Web.Management.TraceWebEventProvider

One method of debugging an ASP.NET application is to use the tracing capability built into the system. Tracing enables you to view details on the request, application state, cookies, the control tree, the form collection, and more. Outputting Web events to the trace output is done via the `TraceWebEventProvider` object. Setting the `TraceWebEventProvider` instance in a configuration file is illustrated in Listing 12-22.

Listing 12-22: Writing Web events to the trace output using TraceWebEventProvider

```
<configuration>
   <system.web>

      <healthMonitoring>

         <!-- Other nodes removed for clarity -->

         <providers>
            <clear />
            <add name="TraceWebEventProvider"
             type="System.Web.Management.TraceWebEventProvider,System.Web"
             maxEventLength="4096"
             maxMessagesPerNotification="1" />
         </providers>

      </healthMonitoring>

   </system.web>
</configuration>
```

Remember, even with the provider in place, you must assign the provider to the particular errors you are wishing to trap. This is accomplished through the `<rules>` section of the health monitoring system. The `IisTraceWebEventProvider` is the same except that the tracing information is sent to IIS rather than the ASP.NET tracing system.

System.Web.Management.WmiWebEventProvider

The last provider built into the health monitoring system is the `WmiWebEventProvider`. This provider enables you to map any Web events that come from the health monitoring system to Windows Management Instrumentation (WMI) events. When passed to the WMI subsystem, you can represent the events as objects. This mapping to WMI events is accomplished through the `aspnet.mof` file found at `C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727`.

By default, the `WmiWebEventProvider` is already set up for you, and you simply need to map the Web events you are interested in to the already-declared `WmiWebEventProvider` in the `<rules>` section of the health monitoring declaration. This declaration is documented in `web.config.comments` file in the CONFIG folder of the Microsoft .NET Framework install on your server. This is illustrated in Listing 12-23 (the `WmiWebEventProvider` is presented in bold).

Listing 12-23: The WmiWebEventProvider definition in the web.config.comments file

```
<configuration>
   <system.web>

      <healthMonitoring>
         <!-- Other nodes removed for clarity -->

         <providers>
            <clear />
            <add name="EventLogProvider"
             type="System.Web.Management.EventLogWebEventProvider,
               System.Web,Version=2.0.0.0,Culture=neutral,
               PublicKeyToken=b03f5f7f11d50a3a" />
            <add connectionStringName="LocalSqlServer"
             maxEventDetailsLength="1073741823" buffer="false"
             bufferMode="Notification" name="SqlWebEventProvider"
             type="System.Web.Management.SqlWebEventProvider,
               System.Web,Version=2.0.0.0,Culture=neutral,
               PublicKeyToken=b03f5f7f11d50a3a" />
            <add name="WmiWebEventProvider"
             type="System.Web.Management.WmiWebEventProvider,
               System.Web,Version=2.0.0.0,Culture=neutral,
               PublicKeyToken=b03f5f7f11d50a3a" />
         </providers>

      </healthMonitoring>

   </system.web>
</configuration>
```

Remember, the wonderful thing about how the health monitoring system uses the provider model is that it permits more than a single provider for the Web events that the system traps.

Configuration Providers

A wonderful feature of ASP.NET 3.5 is that it enables you to actually encrypt sections of your configuration files. You are able to encrypt defined ASP.NET sections of the `web.config` file as well as custom sections that you have placed in the file yourself. This is an ideal way of keeping sensitive configuration information away from the eyes of everyone who peruses the file repository of your application.

By default, ASP.NET 3.5 provides two possible configuration providers out of the box. These providers are defined in the following list:

- `System.Configuration.DpapiProtectedConfigurationProvider`: Provides you with the capability to encrypt and decrypt configuration sections using the data protection API (DPAPI) that is built into the Windows operating system.

- `System.Configuration.RsaProtectedConfigurationProvider`: Provides you with the capability to encrypt and decrypt configuration sections using an RSA public-key encryption algorithm.

These two providers used for encryption and decryption of the configuration sections inherit from the `ProtectedConfigurationProvider` base class. This is illustrated in Figure 12-15.

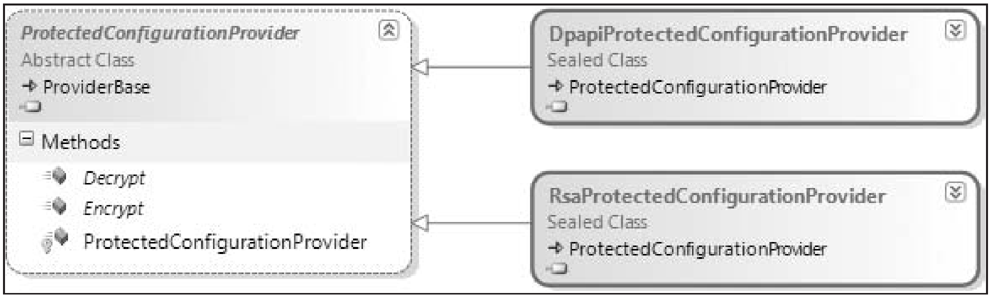

ProtectedConfigurationProvider
Abstract Class
ProviderBase
Methods
Decrypt
Encrypt
ProtectedConfigurationProvider
DpapiProtectedConfigurationProvider
Sealed Class
ProtectedConfigurationProvider
RsaProtectedConfigurationProvider
Sealed Class
ProtectedConfigurationProvider

Figure 12-15

You can find information on how to use these providers to encrypt and decrypt configuration sections in Chapter 32.

Next, you review each of these providers.

System.Configuration.DpapiProtectedConfigurationProvider

The `DpapiProtectedConfigurationProvider` class allows you to encrypt and decrypt configuration sections using the Windows Data Protection API (DPAPI). This provider enables you to perform these encryption and decryption tasks on a per-machine basis. This is not a good provider to use on a Web farm. If you are using protected configuration on your configuration files in a Web farm, you might want to turn your attention to the `RsaProtectedConfigurationProvider`.

If you look in the `machine.config` on your server, you see a definition in place for both the `DpapiProtectedConfigurationProvider` and the `RsaProtectedConfigurationProvider`. The `RsaProtectedConfigurationProvider` is set as the default configuration provider. To establish the `DpapiProtectedConfigurationProvider` as the default provider, you might use the `web.config` file of your application, or you might change the `defaultProvider` attribute in the `machine.config` file for the `<configProtectedData>` node. Changing it in the `web.config` is illustrated in Listing 12-24.

Listing 12-24: Using the DpapiProtectedConfigurationProvider in the web.config

```
<configuration>

  <configProtectedData defaultProvider="DataProtectionConfigurationProvider">
    <providers>
      <clear />
      <add name="DataProtectionConfigurationProvider"
       type="System.Configuration.DpapiProtectedConfigurationProvider,
         System.Configuration, Version=2.0.0.0,
         Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
```

Continued

```
         description="Uses CryptProtectData and CryptUnProtectData Windows
           APIs to encrypt and decrypt"
         useMachineProtection="true"
         keyEntropy="RandomStringValue" />
      </providers>
    </configProtectedData>

  </configuration>
```

The provider is defined within the `<configProtectedData>` section of the configuration file. Note that this configuration section sits *outside* the `< system.web >` section.

The two main attributes of this provider definition are the `useMachineProtection` and the `keyEntropy` attributes.

The `useMachineProtection` attribute by default is set to `true`, meaning that all applications in the server share the same means of encrypting and decrypting configuration sections. This also means that applications residing on the same machine can perform encryption and decryption against each other. Setting the `useMachineProtection` attribute to `false` means that the encryption and decryption are done on an application basis only. This setting also means that you must change the account that the application runs against so it is different from the other applications on the server.

The `keyEntropy` attribute provides a lightweight approach to prevent applications from decrypting each other's configuration sections. The `keyEntropy` attribute can take any random string value to take part in the encryption and decryption processes.

System.Configuration.RsaProtectedConfigurationProvider

The default provider for encrypting and decrypting configuration sections is the `RsaProtectedConfigurationProvider`. You can see this setting in the `machine.config` file on your application server. Code from the `machine.config` file is presented in Listing 12-25.

Listing 12-25: The RsaProtectedConfigurationProvider declaration in the machine.config

```
  <configuration>

    <configProtectedData defaultProvider="RsaProtectedConfigurationProvider">
      <providers>
        <add name="RsaProtectedConfigurationProvider"
         type="System.Configuration.RsaProtectedConfigurationProvider,
           System.Configuration, Version=2.0.0.0, Culture=neutral,
           PublicKeyToken=b03f5f7f11d50a3a"
         description="Uses RsaCryptoServiceProvider to encrypt and decrypt"
         keyContainerName="NetFrameworkConfigurationKey" cspProviderName=""
         useMachineContainer="true" useOAEP="false" />
        <add name="DataProtectionConfigurationProvider"
         type="System.Configuration.DpapiProtectedConfigurationProvider,
           System.Configuration, Version=2.0.0.0, Culture=neutral,
           PublicKeyToken=b03f5f7f11d50a3a"
         description="Uses CryptProtectData and CryptUnProtectData
           Windows APIs to encrypt and decrypt"
         useMachineProtection="true" keyEntropy="" />
      </providers>
    </configProtectedData>

  </configuration>
```

The `RsaProtectedConfigurationProvider` uses Triple-DES encryption to encrypt the specified sections of the configuration file. This provider only has a couple of attributes available to it. These attributes are detailed a bit further on in the chapter.

The `keyContainerName` attribute is the defined key container that is used for the encryption/decryption process. By default, this provider uses the default key container built into the .NET Framework, but you can easily switch an application to another key container via this attribute.

The `cspProviderName` attribute is only used if you have specified a custom cryptographic service provider (CSP) to use with the Windows cryptographic API (CAPI). If so, you specify the name of the CSP as the value of the `cspProviderName` attribute.

The `useMachineContainer` attribute enables you to specify that you want either a machine-wide or user-specific key container. This attribute is quite similar to the `useMachineProtection` attribute found in the `DpapiProtectedConfigurationProvider`.

The `useOAEP` attribute specifies whether to turn on the Optional Asymmetric Encryption and Padding (OAEP) capability when performing the encryption/decryption process. This is set to `false` by default only because Windows 2000 does not support this capability. If your application is being hosted on either Windows Server 2008, Windows Server 2003, or Windows XP, you can change the value of the `useOAEP` attribute to `true`.

The WebParts Provider

Another feature of ASP.NET 3.5 is the capability to build your applications utilizing the new portal framework. The new portal framework provides an outstanding way to build a modular Web site that can be customized with dynamically reapplied settings on a per-user basis. Web Parts are objects in the Portal Framework that the end user can open, close, minimize, maximize, or move from one part of the page to another.

Web parts and the new portal framework are covered in Chapter 17.

The state of these modular components, the Web Parts, must be stored somewhere so they can be reissued on the next visit for the assigned end user. The single provider available for remembering the state of the Web Parts is `System.Web.UI.WebControls.WebParts.SqlPersonalizationProvider`, which provides you with the capability to connect the ASP.NET 3.5 portal framework to Microsoft's SQL Server 2000/2005/2008 as well as to the new Microsoft SQL Server Express Edition.

This single class for the portal framework inherits from the `PersonalizationProvider` base class. This is illustrated in Figure 12-16.

You will find the defined `SqlPersonalizationProvider` in the `web.config` file found in the .NET Framework's configuration folder (`C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG`). This definition is presented in Listing 12-26.

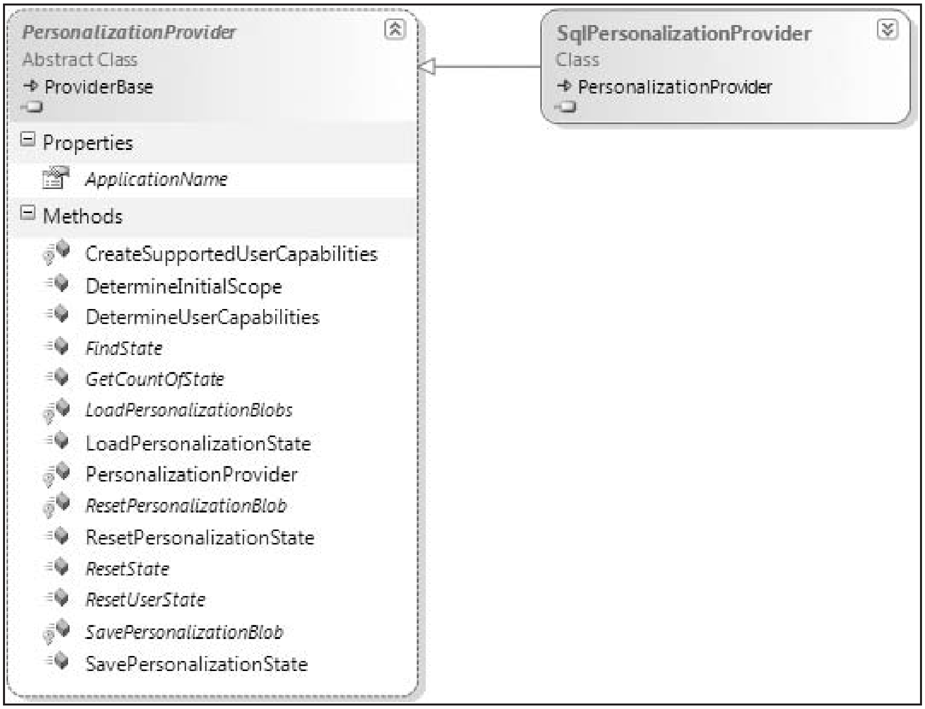

PersonalizationProvider
Abstract Class
ProviderBase
Properties
ApplicationName
Methods
CreateSupportedUserCapabilities
DetermineInitialScope
DetermineUserCapabilities
FindState
GetCountOfState
LoadPersonalizationBlobs
LoadPersonalizationState
PersonalizationProvider
ResetPersonalizationBlob
ResetPersonalizationState
ResetState
ResetUserState
SavePersonalizationBlob
SavePersonalizationState
SqlPersonalizationProvider
Class
PersonalizationProvider

Figure 12-16

Listing 12-26: The SqlPersonalizationProvider definition in the web.config file

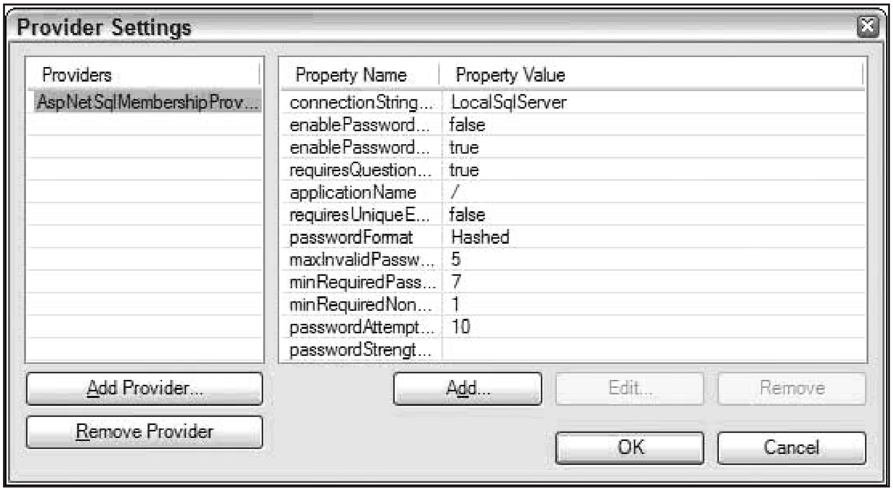
```
<configuration>

   <system.web>

        <webParts>
            <personalization>
                <providers>
                    <add connectionStringName="LocalSqlServer"
                     name="AspNetSqlPersonalizationProvider"
                     type="System.Web.UI.WebControls.WebParts.
                       SqlPersonalizationProvider, System.Web,
                       Version=2.0.0.0, Culture=neutral,
                       PublicKeyToken=b03f5f7f11d50a3a" />
                </providers>

                <authorization>
                    <deny users="*" verbs="enterSharedScope" />
                    <allow users="*" verbs="modifyState" />
                </authorization>
             </personalization>

             <transformers>
                 <add name="RowToFieldTransformer"
                type="System.Web.UI.WebControls.WebParts.RowToFieldTransformer" />
                <add name="RowToParametersTransformer"
                type="System.Web.UI.WebControls.WebParts.
                  RowToParametersTransformer" />
            </transformers>
        </webParts>

   </system.web>

</configuration>
```

As you can see the provider declaration is shown in bold in Listing 12-26. As with the other SQL Server–based providers presented in this chapter, this provider works with SQL Server Express Edition by default. To change this to work with SQL Server 2000, 2005, or 2008, you must make a connection to your database within the `<connectionStrings>` section and make an association to this new connection string in the `SqlPersonalizationProvider` declaration using the `connectionStringName` attribute.

Configuring Providers

As you have seen in this chapter, you can easily associate these systems in ASP.NET 3.5 to a large base of available providers. From there, you can also configure the behavior of the associated providers through the attributes exposed from the providers. This can easily be done through either the system-wide configuration files (such as the `machine.config` file) or through more application-specific configuration files (such as the `web.config` file).

You can also just as easily configure providers through the GUI-based configuration systems such as the ASP.NET Web Site Administration Tool or through the new ASP.NET MMC snap-in. Both of these items are covered in detail in Chapter 32. An example of using the ASP.NET MMC snap-in Windows XP to visually configure a provider is presented in Figure 12-17.



Providers
AspNetSqlMembershipProv...

Property Name	Property Value
connectionString...	LocalSqlServer
enablePassword...	false
enablePassword...	true
requiresQuestion...	true
applicationName	/
requiresUniqueE...	false
passwordFormat	Hashed
maxInvalidPassw...	5
minRequiredPass...	7
minRequiredNon...	1
passwordAttempt...	10
passwordStrengt...	

Figure 12-17

From this figure, you can see that you can add and remove providers in the membership system of your application. You can also change the values assigned to individual attributes directly in the GUI.

Summary

This chapter covered the basics of the provider model and what providers are available to you as you start working with the various ASP.NET systems at your disposal. It is important to understand the built-in providers available for each of these systems and how you can fine-tune the behaviors of each provider.

This provider model allows for an additional level of abstraction and permits you to decide for yourself on the underlying data stores to be used for the various systems. For instance, you have the power to decide whether to store the membership and role management information in SQL Server or in Oracle without making any changes to business or presentation logic!

The next chapter shows how to take the provider model to the next level.

13

Extending the Provider Model

The last chapter introduced the provider model found in ASP.NET 3.5 and explained how it is used with the membership and role management systems.

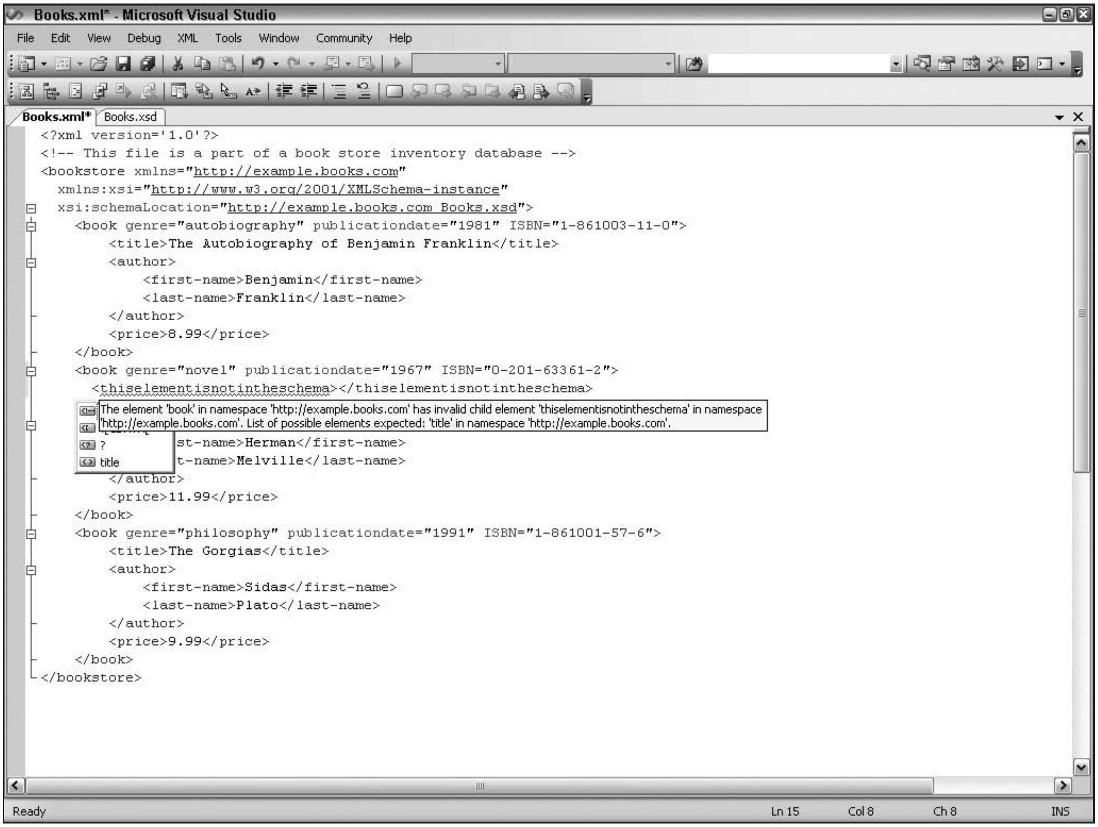
As discussed in the previous chapter, these systems in ASP.NET 3.5 require that some type of user state be maintained for long periods of time. Their time-interval and security requirements for state storage are greater than those for earlier systems that simply used the `Session` object. Out of the box, ASP.NET 3.5 gives you a series of providers to use as the underlying connectors for any data storage needs that arise from state management for these systems.

The providers that come with the default install of the .NET Framework 3.5 include the most common means of state management data storage needed to work with any of the systems. But like most things in .NET, you can customize and extend the providers that are supplied.

This chapter looks at some of the ways to extend the provider model found in ASP.NET 3.5. This chapter also reviews a couple of sample extensions to the provider model. First, however, you look at some of the simpler ways to modify and extend the providers already present in the default install of .NET 3.5.

Providers Are One Tier in a Larger Architecture

Remember from the previous chapter that providers allow you to define the data-access tier for many of the systems in ASP.NET 3.5. They also enable you to define your core business logic implementation on how the data is manipulated or handled. They enable you to use the various controls and APIs that compose these systems in a uniform manner regardless of the underlying data storage method of the provider. The provider model also allows you to easily swap one provider for another without affecting the underlying controls and API that are interacting with the provider. This model is presented in Figure 13-1.


Books.xml* - Microsoft Visual Studio
File Edit View Debug XML Tools Window Community Help
Books.xml*
Books.xsd
?
title
Ready
Ln 15
Col 8
Ch 8
INS

```
<?xml version='1.0'?>
<!-- This file is a part of a book store inventory database -->
<bookstore xmlns="http://example.books.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://example.books.com Books.xsd">
    <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
        <title>The Autobiography of Benjamin Franklin</title>
        <author>
            <first-name>Benjamin</first-name>
            <last-name>Franklin</last-name>
        </author>
        <price>8.99</price>
    </book>
    <book genre="novel" publicationdate="1967" ISBN="0-201-63361-2">
      <thiselementisnotintheschema></thiselementisnotintheschema>

                st-name>Herman</first-name>
                t-name>Melville</last-name>
        </author>
        <price>11.99</price>
    </book>
    <book genre="philosophy" publicationdate="1991" ISBN="1-861001-57-6">
        <title>The Gorgias</title>
        <author>
            <first-name>Sidas</first-name>
            <last-name>Plato</last-name>
        </author>
        <price>9.99</price>
    </book>
</bookstore>
```

The element 'book' in namespace 'http://example.books.com' has invalid child element 'thiselementisnotintheschema' in namespace 'http://example.books.com'. List of possible elements expected: 'title' in namespace 'http://example.books.com'.

Figure 13-1

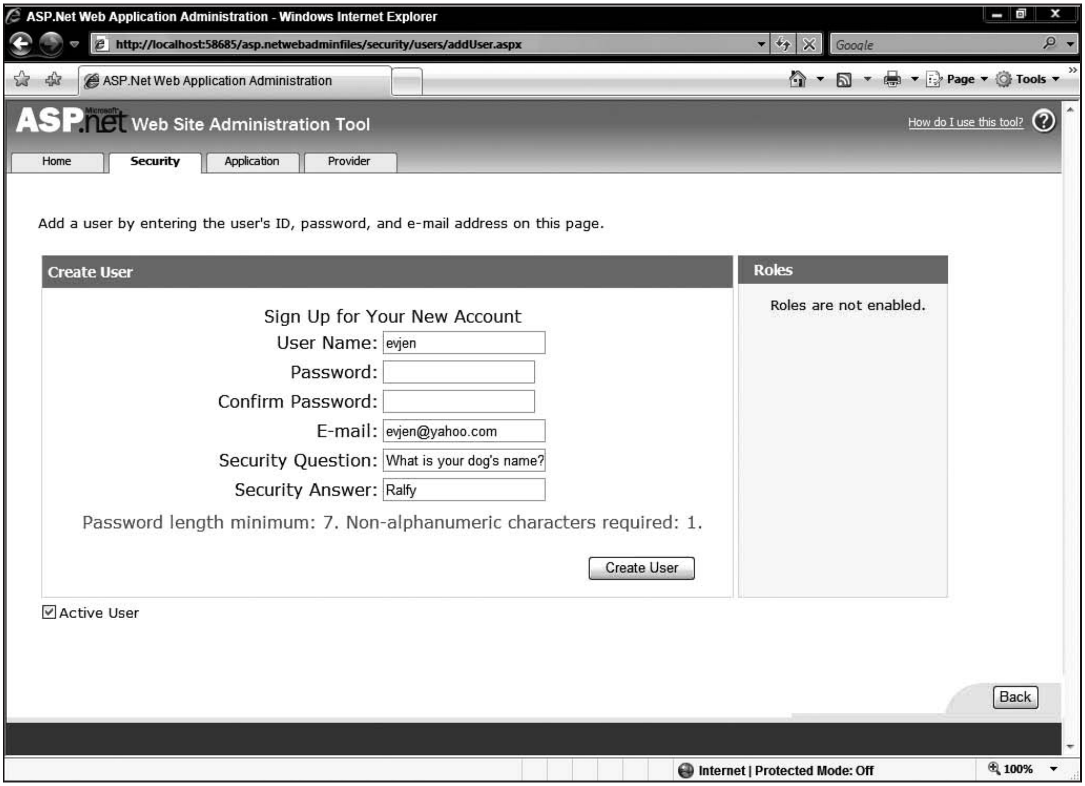
From this diagram, you can see that both the controls utilized in the membership system, as well as the Membership API, use the defined provider. Changing the underlying provider does not change the controls or the API, but you can definitely modify how these items behave (as you will see shortly). You can also simply change the location where the state management required by these items is stored. Changing the underlying provider, in this case, does not produce any change whatsoever in the controls or the API; instead, their state management data points are simply rerouted to another data store type.

Modifying Through Attribute-Based Programming

Probably the easiest way to modify the behaviors of the providers built into the .NET Framework 3.5 is through attribute-based programming. In ASP.NET 3.5, you can apply quite advanced behavior modification through attribute usage. You can apply both the server controls and the settings in the various application configuration files. Using the definitions of the providers found in either the `machine.config` files or within the root `web.config` file, you can really change provider behavior. This chapter gives you an example of how to modify the `SqlMembershipProvider`.

Simpler Password Structures Through the SqlMembershipProvider

When you create users with the `SqlMembershipProvider` instance, whether you are using SQL Server Express or Microsoft's SQL Server 2000/2005/2008, notice that the password required to create a user is a semi-strong password. This is evident when you create a user through the ASP.NET Web Site Administration Tool, as illustrated in Figure 13-2.


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:58685/asp.netwebadminfiles/security/users/addUser.aspx
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Create User
Roles
Create User
Back
Internet | Protected Mode: Off
100%

Add a user by entering the user's ID, password, and e-mail address on this page.

Sign Up for Your New Account

User Name: evjen

Password:

Confirm Password:

E-mail: evjen@yahoo.com

Security Question: What is your dog's name?

Security Answer: Ralfy

Password length minimum: 7. Non-alphanumeric characters required: 1.

Roles are not enabled.

☑ Active User

Figure 13-2

On this screen, I attempted to enter a password, and was notified that the password did not meet the application's requirements. Instead, was warned that the minimum password length is seven characters and that at least one non-alphanumeric character is required. This means that a password such as *Bubbles!* is what is required. This kind of behavior is specified by the membership provider and not by the controls or the API used in the membership system. You find the definition of the requirements in the `machine.config.comments` file located at `C:\WINDOWS\Microsoft.NEt\Framework\v20.50727\CONFIG`. This definition is presented in Listing 13-1.

Listing 13-1: The SqlMembershipProvider instance declaration

```
<configuration>
   <system.web>
        <membership defaultProvider="AspNetSqlMembershipProvider"
         userIsOnlineTimeWindow="15" hashAlgorithmType="" >
            <providers>
                <clear />
                <add connectionStringName="LocalSqlServer"
                 enablePasswordRetrieval="false"
                 enablePasswordReset="true"
                 requiresQuestionAndAnswer="true"
                 applicationName="/"
                 requiresUniqueEmail="false"
                 passwordFormat="Hashed"
                 maxInvalidPasswordAttempts="5"
                 minRequiredPasswordLength="7"
                 minRequiredNonalphanumericCharacters="1"
                 passwordAttemptWindow="10"
                 passwordStrengthRegularExpression=""
                 name="AspNetSqlMembershipProvider"
                 type="System.Web.Security.SqlMembershipProvider,
                    System.Web, Version=2.0.0.0, Culture=neutral,
                    PublicKeyToken=b03f5f7f11d50a3a" />
            </providers>
        </membership>
   </system.web>
</configuration>
```

Looking over the attributes of this provider, notice the `minRequiredPasswordLength` and the `minRequiredNonalphanumericCharacters` attributes define this behavior. To change this behavior across every application on the server, you simply change these values in this file. the author would, however, suggest simply changing these values in your application's `web.config` file, as shown in Listing 13-2.

Listing 13-2: Changing attribute values in the web.config

```
<configuration>

   <system.web>

        <authentication mode="Forms" />

        <membership>
          <providers>
            <clear />
            <add name="AspNetSqlMembershipProvider"
             type="System.Web.Security.SqlMembershipProvider,
               System.Web, Version=2.0.0.0, Culture=neutral,
               PublicKeyToken=b03f5f7f11d50a3a"
             connectionStringName="LocalSqlServer"
             enablePasswordRetrieval="false"
             enablePasswordReset="true"
             requiresQuestionAndAnswer="true"
```

```
                requiresUniqueEmail="false"
                passwordFormat="Hashed"
                maxInvalidPasswordAttempts="5"</b>
                minRequiredPasswordLength="4"
                minRequiredNonalphanumericCharacters="0"
                passwordAttemptWindow="10" />
            </providers>
          </membership>
     </system.web>

  </configuration>
```

In this example, the password requirements are changed through the `minRequiredPasswordLength` and `minRequiredNonalphanumericCharacters` attributes. In this case, the minimum length allowed for a password is four characters, and none of those characters is required to be non-alphanumeric (for example, a special character such as `!`, `$`, or `#`).

Redefining a provider in the application's `web.config` is a fairly simple process. In the example in Listing 13-2, you can see that the `<membership>` element is quite similar to the same element presented in the `machine.config` file.

You have a couple of options when defining your own instance of the `SqlMembershipProvider`. One approach, as presented in Listing 13-2, is to redefine the named instance of the `SqlMembershipProvider` that is defined in the `machine.config` file (`AspNetSqlMembershipProvider`, the value from the name attribute in the provider declaration). If you take this approach, you must *clear* the previous defined instance of `AspNetSqlMembershipProvider`. You must redefine the `AspNetSqlMembershipProvider` using the `< clear / >` node within the `<providers>` section. Failure to do so causes an error to be thrown stating that this provider name is already defined.

After you have cleared the previous instance of `AspNetSqlMembershipProvider`, you redefine this provider using the `<add>` element. In the case of Listing 13-2, you can see that the password requirements are redefined with the use of new values for the `minRequiredPasswordLength` and the `minRequiredNonalphanumericCharacters` attributes (shown in bold).

The other approach to defining your own instance of the `SqlMembershipProvider` is to give the provider defined in the `<add>` element a unique value for the `name` attribute. If you take this approach, you must specify this new named instance as the default provider of the membership system using the `defaultProvider` attribute. This approach is presented in Listing 13-3.

Listing 13-3: Defining your own named instance of the SqlMembershipProvider

```
<membership defaultProvider="MyVeryOwnAspNetSqlMembershipProvider">
   <providers>
      <add name="MyVeryOwnAspNetSqlMembershipProvider"
       type="System.Web.Security.SqlMembershipProvider,
         System.Web, Version=2.0.0.0, Culture=neutral,
         PublicKeyToken=b03f5f7f11d50a3a"
       connectionStringName="LocalSqlServer"
       enablePasswordRetrieval="false"
       enablePasswordReset="true"
       requiresQuestionAndAnswer="true"
```

Continued

```
            requiresUniqueEmail="false"
            passwordFormat="Hashed"
            maxInvalidPasswordAttempts="5"
            minRequiredPasswordLength="4"
            minRequiredNonalphanumericCharacters="0"
            passwordAttemptWindow="10" />
      </providers>
   </membership>
```

In this case, the `SqlMembershipProvider` instance in the `machine.config` file (defined under the `AspNetSqlMembershipProvider` name) is not even redefined. Instead, a completely new named instance (`MyVeryOwnAspNetSqlMembershipProvider)` is defined here in the `web.config`.

Stronger Password Structures Through the SqlMembershipProvider

Next, this chapter shows you how to actually make the password structures a little more complicated. You can, of course, accomplish this task in a couple of ways. One approach is to use the same `minRequiredPasswordLength` and `minRequiredNonalphanumericCharacters` attributes (as shown earlier) to make the password meet a required length (longer passwords usually mean more secure passwords) and to make the password contain a certain number of non-alphanumeric characters (this also makes for a more secure password).

Another option is to use the `passwordStrengthRegularExpression` attribute. If the `minRequiredPasswordLength` and the `minRequiredNonalphanumericCharacters` attributes cannot give you the password structure you are searching for, then using the `passwordStrengthRegularExpression` attribute is your next best alternative.

For an example of using this attribute, suppose you require that the user's password is his or her U.S. Social Security number. You can then define your provider as shown in Listing 13-4.

Listing 13-4: A provider instance in the web.config to change the password structure

```
<configuration>

   <system.web>

       <authentication mode="Forms" />

       <membership>
         <providers>
           <clear />
           <add name="AspNetSqlMembershipProvider"
            type="System.Web.Security.SqlMembershipProvider,
              System.Web, Version=2.0.0.0, Culture=neutral,
              PublicKeyToken=b03f5f7f11d50a3a"
            connectionStringName="LocalSqlServer"
            enablePasswordRetrieval="false"
            enablePasswordReset="true"
            requiresQuestionAndAnswer="true"
```

```
             requiresUniqueEmail="false"
             passwordFormat="Hashed"
             maxInvalidPasswordAttempts="5"
             passwordAttemptWindow="10"
             passwordStrengthRegularExpression="\d{3}-\d{2}-\d{4}" />
          </providers>
        </membership>

   </system.web>

</configuration>
```

Instead of using the `minRequiredPasswordLength` and the `minRequiredNonalphanumericCharacters` attributes, the `passwordStrengthRegularExpression` attribute is used and given a value of `\d{3}–\d{2}–\d{4}`. This regular expression means that the password should have three digits followed by a dash or hyphen, followed by two digits and another dash or hyphen, finally followed by four digits.

The lesson here is that you have many ways to modify the behaviors of the providers already available in the .NET Framework 3.5 install. You can adapt a number of providers built into the framework to suit your needs by using attribute-based programming. The `SqlMembershipProvider` example demonstrated this, and you can just as easily make similar types of modifications to any of the other providers.

Examining ProviderBase

All the providers derive in some fashion from the class, `ProviderBase`, found in the `System.Configuration.Provider` namespace. `ProviderBase` is an abstract class used to define a base template for inheriting providers. Looking at `ProviderBase`, note that there isn't much to this abstract class, as illustrated in Figure 13-3.

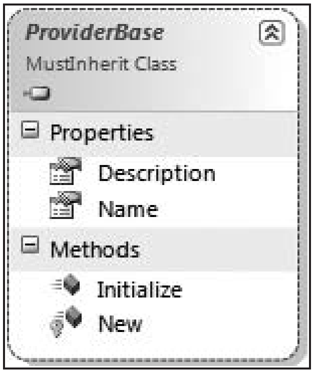


Figure 13-3

As stated, there is not much to this class. It is really just a root class for a provider that exists to allow providers to initialize themselves.

The `Name` property is used to provide a friendly name, such as AspNetSqlRoleProvider. The `Description` property is used to enable a textual description of the provider, which can then be used later by any administration tools. The main item in the `ProviderBase` class is the `Initialize()` method. The constructor for `Initialize()` is presented here:

```
public virtual void Initialize(string name,
   System.Collections.Specialized.NameValueCollection config);
```

Note the two parameters to the `Initialize()` method. The first is the `name` parameter, which is simply the value assigned to the `name` attribute in the provider declaration in the configuration file. The `config` parameter is of type `NameValueCollection`, which is a collection of name/value pairs. These name/value pairs are the items that are also defined in the provider declaration in the configuration file as all the various attributes and their associated values.

When looking over the providers that are included in the default install of ASP.NET 3.5, note that each of the providers has defined a class you can derive from that implements the `ProviderBase` abstract class. For instance, looking at the model in place for the membership system, you can see a base `MembershipProvider` instance that is inherited in the final `SqlMembershipProvider` declaration. The `MembershipProvider`, however, implements `ProviderBase` itself. This model is presented in Figure 13-4.

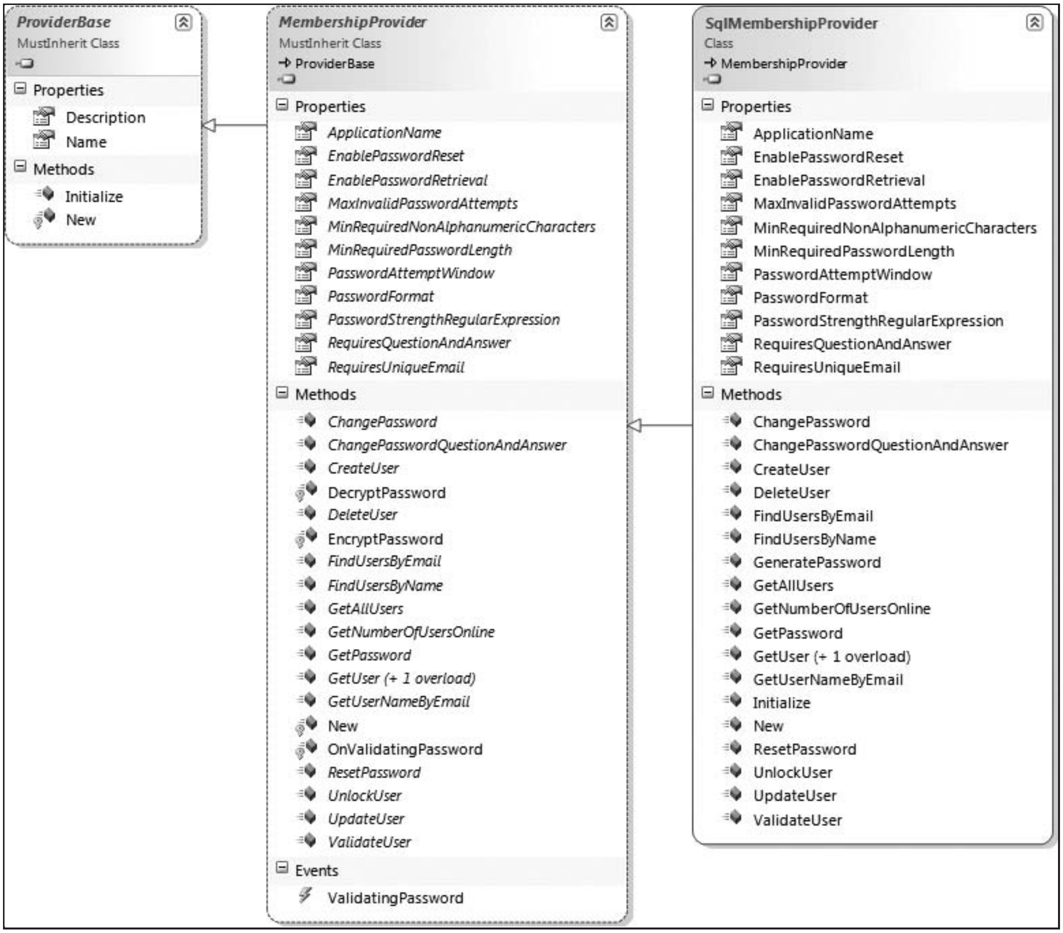

ProviderBase
MustInherit Class
Properties
Description
Name
Methods
Initialize
New
MembershipProvider
MustInherit Class
ProviderBase
Properties
ApplicationName
EnablePasswordReset
EnablePasswordRetrieval
MaxInvalidPasswordAttempts
MinRequiredNonAlphanumericCharacters
MinRequiredPasswordLength
PasswordAttemptWindow
PasswordFormat
PasswordStrengthRegularExpression
RequiresQuestionAndAnswer
RequiresUniqueEmail
Methods
ChangePassword
ChangePasswordQuestionAndAnswer
CreateUser
DecryptPassword
DeleteUser
EncryptPassword
FindUsersByEmail
FindUsersByName
GetAllUsers
GetNumberOfUsersOnline
GetPassword
GetUser (+ 1 overload)
GetUserNameByEmail
New
OnValidatingPassword
ResetPassword
UnlockUser
UpdateUser
ValidateUser
Events
ValidatingPassword
SqlMembershipProvider
Class
MembershipProvider
Properties
ApplicationName
EnablePasswordReset
EnablePasswordRetrieval
MaxInvalidPasswordAttempts
MinRequiredNonAlphanumericCharacters
MinRequiredPasswordLength
PasswordAttemptWindow
PasswordFormat
PasswordStrengthRegularExpression
RequiresQuestionAndAnswer
RequiresUniqueEmail
Methods
ChangePassword
ChangePasswordQuestionAndAnswer
CreateUser
DeleteUser
FindUsersByEmail
FindUsersByName
GeneratePassword
GetAllUsers
GetNumberOfUsersOnline
GetPassword
GetUser (+ 1 overload)
GetUserNameByEmail
Initialize
New
ResetPassword
UnlockUser
UpdateUser
ValidateUser

Figure 13-4

Notice that each of the various systems has a specific base provider implementation for you to work with. There really cannot be a single provider that addresses the needs of all the available systems. Looking at Figure 13-4, you can see that the `MembershipProvider` instance exposes some very specific functionality required by the ASP.NET membership system. The methods exposed are definitely not needed by the role management system or the Web parts capability.

With these various base implementations in place, when you are creating your own customizations for working with the ASP.NET membership system, you have a couple of options available to you. First, you can simply create your own provider directly implementing the `ProviderBase` class and working from the ground up. I do not recommend this approach, however, because abstract classes are already in place for you to use with the various systems. So, as I mentioned, you can just implement the `MembershipProvider` instance (a better approach) and work from the model it provides. Finally, if you are working with SQL Server in some capacity and simply want to change the underlying behaviors of this provider, you can inherit from `SqlMembershipProvider` and modify the behavior of the class from this inheritance. Next, this chapter covers the various means of extending the provider model through examples.

Building Your Own Providers

You now examine the process of building your own provider to use within your ASP.NET 3.5 application. Actually, providers are not that difficult to put together (as you will see shortly) and can even be created directly in any of your ASP.NET 3.5 projects. The example demonstrates building a membership provider that works from an XML file. For a smaller Web site, this might be a common scenario. For larger Web sites and Web-based applications, you probably want to use a database of some kind, rather than an XML file, for managing users.

You have a couple of options when building you own membership provider. You can derive from a couple of classes, the `SqlMembershipProvider` class, or the `MembershipProvider` class, to build the functionality you need. You derive from the `SqlMembershipProvider` class only if you want to extend or change the behavior of the membership system as it interacts with SQL. Because the goal here is to build a read-only XML membership provider, deriving from this class is inappropriate. In this case, it is best to base everything on the `MembershipProvider` class.

Creating the CustomProviders Application

For this example, create a new Web site project called CustomProviders in the language of your choice. For this example, you want to build the new membership provider directly in the Web application itself. Another option is to build the provider in a Class Library project and then to reference the generated DLL in your Web project. Either way is fine in the end.

Because you are going to build this directly in the Web site project itself, you create the App_Code folder in your application. This is the location you want to place the class file that you create. The class file is the actual provider in this case.

After the App_Code folder is in place, create a new class in this folder and call the class either `XmlMembershipProvider.vb` or `XmlMembershipProvider.cs`, depending on the language you are using. With this class now in place, have your new `XmlMembershipProvider` class derive from `MembershipProvider`. To accomplish this and to know which methods and properties to override, you can use Visual Studio

2008 to build a skeleton of the class you want to create. You can step through this process starting with the code demonstrated in Listing 13-5.

Listing 13-5: The start of your XmlMembershipProvider class

VB

```
Imports Microsoft.VisualBasic
Imports System.Xml
Imports System.Configuration.Provider
Imports System.Web.Hosting
Imports System.Collections
Imports System.Collections.Generic

Public Class XmlMembershipProvider
    Inherits MembershipProvider

End Class
```

C#

```
using System;
using System.Web.Hosting;
using System.Web.Security;
using System.Xml;
using System.Collections.Generic;

/// <summary>
/// Summary description for XmlMembershipProvider
/// </summary>
public class XmlMembershipProvider: MembershipProvider
{
 public XmlMembershipProvider()
 {
          //
         // TODO: Add constructor logic here
         //
 }
}
```

You make only a few changes to the basic class, `XmlMembershipProvider`. First, you can see that some extra namespaces are imported into the file. This is done so you can later take advantage of .NET's XML capabilities, generics, and more. Also, notice that this new class, the `XmlMembershipProvider` class, inherits from `MembershipProvider`.

Constructing the Class Skeleton Required

In order to get Visual Studio 2008 to build your class with the appropriate methods and properties, take the following steps (depending on the language you are using). If you are using Visual Basic, all you have to do is press the Enter key. In C#, you first place the cursor on the `MembershipProvider` instance in the document window and then select Edit➪IntelliSense➪Implement Abstract Class from the Visual Studio menu. After you perform one of these operations, you see the full skeleton of the class in the document window of Visual Studio. Listing 13-6 shows the code that is generated if you are creating a Visual Basic `XmlMembershipProvider` class.

Listing 13-6: Code generated for the XmlMembershipProvider class by Visual Studio

VB (only)

```
Imports Microsoft.VisualBasic
Imports System.Xml
Imports System.Configuration.Provider
Imports System.Web.Hosting
Imports System.Collections
Imports System.Collections.Generic

Public Class XmlMembershipProvider
    Inherits MembershipProvider

    Public Overrides Property ApplicationName() As String
        Get

        End Get
        Set(ByVal value As String)

        End Set
    End Property

    Public Overrides Function ChangePassword(ByVal username As String, _
      ByVal oldPassword As String, ByVal newPassword As String) As Boolean

    End Function

    Public Overrides Function ChangePasswordQuestionAndAnswer(ByVal username _
      As String, ByVal password As String, ByVal newPasswordQuestion As String, _
      ByVal newPasswordAnswer As String) As Boolean

    End Function

    Public Overrides Function CreateUser(ByVal username As String, _
      ByVal password As String, ByVal email As String, _
      ByVal passwordQuestion As String, ByVal passwordAnswer As String, _
      ByVal isApproved As Boolean, ByVal providerUserKey As Object, _
      ByRef status As System.Web.Security.MembershipCreateStatus) As _
      System.Web.Security.MembershipUser

    End Function

    Public Overrides Function DeleteUser(ByVal username As String, _
      ByVal deleteAllRelatedData As Boolean) As Boolean

    End Function

    Public Overrides ReadOnly Property EnablePasswordReset() As Boolean
        Get

        End Get
    End Property
```

Continued

```
Public Overrides ReadOnly Property EnablePasswordRetrieval() As Boolean
    Get

    End Get
End Property

Public Overrides Function FindUsersByEmail(ByVal emailToMatch As String, _
  ByVal pageIndex As Integer, ByVal pageSize As Integer, _
  ByRef totalRecords As Integer) As _
  System.Web.Security.MembershipUserCollection
End Function

Public Overrides Function FindUsersByName(ByVal usernameToMatch As String, _
  ByVal pageIndex As Integer, ByVal pageSize As Integer, _
  ByRef totalRecords As Integer) As _
  System.Web.Security.MembershipUserCollection

End Function

Public Overrides Function GetAllUsers(ByVal pageIndex As Integer, _
  ByVal pageSize As Integer, ByRef totalRecords As Integer) As _
  System.Web.Security.MembershipUserCollection

End Function

Public Overrides Function GetNumberOfUsersOnline() As Integer

End Function

Public Overrides Function GetPassword(ByVal username As String, _
  ByVal answer As String) As String

End Function

Public Overloads Overrides Function GetUser(ByVal providerUserKey As Object, _
  ByVal userIsOnline As Boolean) As System.Web.Security.MembershipUser

End Function

Public Overloads Overrides Function GetUser(ByVal username As String, _
  ByVal userIsOnline As Boolean) As System.Web.Security.MembershipUser

End Function

Public Overrides Function GetUserNameByEmail(ByVal email As String) As String

End Function

Public Overrides ReadOnly Property MaxInvalidPasswordAttempts() As Integer
    Get

    End Get
End Property
```

Continued

```
Public Overrides ReadOnly Property MinRequiredNonAlphanumericCharacters() _
  As Integer
    Get

    End Get
End Property

Public Overrides ReadOnly Property MinRequiredPasswordLength() As Integer
    Get

    End Get
End Property

Public Overrides ReadOnly Property PasswordAttemptWindow() As Integer
    Get
    End Get
End Property

Public Overrides ReadOnly Property PasswordFormat() As _
  System.Web.Security.MembershipPasswordFormat
    Get

    End Get
End Property
Public Overrides ReadOnly Property PasswordStrengthRegularExpression() As _
  String
    Get

    End Get
End Property

Public Overrides ReadOnly Property RequiresQuestionAndAnswer() As Boolean
    Get
    End Get
End Property

Public Overrides ReadOnly Property RequiresUniqueEmail() As Boolean
    Get

    End Get
End Property

Public Overrides Function ResetPassword(ByVal username As String, _
  ByVal answer As String) As String

End Function

Public Overrides Function UnlockUser(ByVal userName As String) As Boolean

End Function

Public Overrides Sub UpdateUser(ByVal user As _
  System.Web.Security.MembershipUser)
```

Continued

```
        End Sub

        Public Overrides Function ValidateUser(ByVal username As String, _
          ByVal password As String) As Boolean

        End Function
    End Class
```

Wow, that's a lot of code! Although the skeleton is in place, the next step is to build some of the items that will be utilized by the provider that Visual Studio laid out for you – starting with the XML file that holds all the users allowed to access the application.

Creating the XML User Data Store

Because this is an XML membership provider, the intent is to read the user information from an XML file rather than from a database such as SQL Server. For this reason, you must define the XML file structure that the provider can make use of. The structure that we are using for this example is illustrated in Listing 13-7.

Listing 13-7: The XML file used to store usernames and passwords

```
<?xml version="1.0" encoding="utf-8" ?>
<Users>
  <User>
    <Username>BillEvjen</Username>
    <Password>Bubbles</Password>
    <Email>evjen@yahoo.com</Email>
    <DateCreated>11/10/2008</DateCreated>
  </User>
  <User>
    <Username>ScottHanselman</Username>
    <Password>YabbaDabbaDo</Password>
    <Email>123@msn.com</Email>
    <DateCreated>10/20/2008</DateCreated>
  </User>
  <User>
    <Username>DevinRader</Username>
    <Password>BamBam</Password>
    <Email>456@msn.com</Email>
    <DateCreated>9/23/2008</DateCreated>
  </User>
</Users>
```

This XML file holds only three user instances, all of which include the username, password, e-mail address, and the date on which the user is created. Because this is a data file, you should place this file in the `App_Data` folder of your ASP.NET application. You can name the file anything you want; but in this case, we have named the file `UserDatabase.xml`.

Later, this chapter reviews how to grab these values from the XML file when validating users.

Defining the Provider Instance in the web.config File

As you have seen in the last chapter on providers, you define a provider and its behavior in a configuration file (such as the `machine.config` or the `web.config` file). Because this provider is being built for a single application instance, this example defines the provider in the `web.config` file of the application.

The default provider is the `SqlMembershipProvider`, and this is defined in the `machine.config` file on the server. For this example, you must override this setting and establish a new default provider. The XML membership provider declaration in the `web.config` should appear as shown in Listing 13-8.

Listing 13-8: Defining the XmlMembershipProvider in the web.config file

```
<configuration>
   <system.web>

      <authentication mode="Forms"/>

      <membership defaultProvider="XmlFileProvider">
         <providers>
            <add name="XmlFileProvider" type="XmlMembershipProvider"
             xmlUserDatabaseFile="~/App_Data/UserDatabase.xml"/>
         </providers>
      </membership>

   </system.web>
</configuration>
```

In this listing, you can see that the default provider is defined as the `XmlFileProvider`. Because this provider name will not be found in any of the parent configuration files, you must define `XmlFileProvider` in the `web.config` file.

Using the `defaultProvider` attribute, you can define the name of the provider you want to use for the membership system. In this case, it is `XmlFileProvider`. Then you define the `XmlFileProvider` instance using the `<add>` element within the `<providers>` section. The `<add>` element gives a name for the provider — `XmlFileProvider`. It also points to the class (or type) of the provider. In this case, it is the skeleton class you just created — `XmlMembershipProvider`. These are the two most important attributes.

Beyond this, you can create any attribute in your provider declaration that you wish. Whatever type of provider you create, however, you must address the attributes in your provider and act upon the values that are provided with the attributes. In the case of the simple `XmlMembershipProvider`, only a single custom attribute exists — `xmlUserDatabaseFile`. This attribute points to the location of the user database XML file. For this provider, it is an optional attribute. If you do not provide a value for `xmlUserDatabaseFile`, you have a default value. In Listing 13-8, however, you can see that a value is indeed provided for the XML file to use. Note that the `xmlUserDatabaseFile` is simply the filename and nothing more.

One attribute is not shown in the example, but is an allowable attribute because it is addressed in the `XmlMemberhipProvider` class. This attribute, the `applicationName` attribute, points to the application that the `XmlMembershipProvider` instance should address. The default value, which you can also place in this provider declaration within the configuration file, is illustrated here:

```
applicationName="/"
```

Not Implementing Methods and Properties of the MembershipProvider Class

Now turn your attention to the `XmlMembershipProvider` class. The next step is to implement any methods or properties needed by the provider. You are not required to make any *real* use of the methods contained in this skeleton; instead, you can simply build-out only the methods you are interested in working with. For instance, if you do not allow for programmatic access to change passwords (and, in turn, the controls that use this programmatic access), you either want not to initiate an action or to throw an exception if someone tries to implement this method. This is illustrated in Listing 13-9.

Listing 13-9: Not implementing one of the available methods by throwing an exception

VB

```
Public Overrides Function ChangePassword(ByVal username As String, _
  ByVal oldPassword As String, ByVal newPassword As String) As Boolean
        Throw New NotSupportedException()
End Function
```

C#

```
public override bool ChangePassword(string username,
  string oldPassword, string newPassword)
{
   throw new NotSupportedException();
}
```

In this case, a `NotSupportedException` is thrown if the `ChangePassword()` method is invoked. If you do not want to throw an actual exception, you can simply return a `false` value and not take any other action, as shown in Listing 13-10 (although this might annoy a developer who is trying to implement this and does not understand the underlying logic of the method).

Listing 13-10: Not implementing one of the available methods by returning a false value

VB

```
Public Overrides Function ChangePassword(ByVal username As String, _
  ByVal oldPassword As String, ByVal newPassword As String) As Boolean
        Return False
End Function
```

C#

```
public override bool ChangePassword(string username,
  string oldPassword, string newPassword)
{
   return false;
}
```

This chapter does not address every possible action you can take with `XmlMembershipProvider` and, therefore, you may want to work through the available methods and properties of the derived `MembershipProvider` instance and make the necessary changes to any items that you won't be using.

Implementing Methods and Properties of the MembershipProvider Class

Now it is time to implement some of the methods and properties available from the `MembershipProvider` class in order to get the `XmlMembershipProvider` class to work. The first items are some private variables that can be utilized by multiple methods throughout the class. These variable declarations are presented in Listing 13-11.

Listing 13-11: Declaring some private variables in the XmlMembershipProvider class

VB

```
Public Class XmlMembershipProvider
    Inherits MembershipProvider

    Private _AppName As String
    Private _MyUsers As Dictionary(Of String, MembershipUser)
    Private _FileName As String

    ' Code removed for clarity

End Class
```

C#

```
public class XmlMembershipProvider : MembershipProvider
{
    private string _AppName;
    private Dictionary<string, MembershipUser> _MyUsers;
    private string _FileName;

   ' Code removed for clarity

}
```

The variables being declared are items needed by multiple methods in the class. The `_AppName` variable defines the application using the XML membership provider. In all cases, it is the local application. You also want to place all the members found in the XML file into a collection of some type. This example uses a dictionary generic type named `_MyUsers`. Finally, this example points to the file to use with the `_FileName` variable.

The ApplicationName Property

After the private variables are in place, the next step is to define the `ApplicationName` property. You now make use of the first private variable — `AppName`. The property definition of `ApplicationName` is presented in Listing 13-12.

Listing 13-12: Defining the ApplicationName property

VB

```
Public Overrides Property ApplicationName() As String
   Get
      Return _AppName
   End Get
   Set(ByVal value As String)
      _AppName = value
   End Set
End Property
```

C#

```
public override string ApplicationName
{
   get
   {
      return _AppName;
   }
   set
   {
      _AppName = value;
   }
}
```

Now that the `ApplicationName` property is defined and in place, you next retrieve the values defined in the `web.config` file's provider declaration (`XmlFileProvider`).

Extending the Initialize() Method

You now extend the `Initialize()` method so that it reads in the custom attribute and its associated values as defined in the provider declaration in the `web.config` file. Look through the class skeleton of your `XmlMembershipProvider` class, and note that no `Initialize()` method is included in the list of available items.

The `Initialize()` method is invoked when the provider is first initialized. It is not a requirement to override this method and, therefore, you won't see it in the declaration of the class skeleton. To put the `Initialize()` method in place within the `XmlMembershipProvider` class, simply type `Public Overrides` (for Visual Basic) or `public override` (for C#) in the class. You are then presented with the `Initialize()` method via IntelliSense, as shown in Figure 13-5.

Placing the `Initialize()` method in your class in this manner is quite easy. Select the `Initialize()` method from the list in IntelliSense and press the Enter key. This gives you a base construction of the method in your code. This is shown in Listing 13-13.

Listing 13-13: The beginnings of the Initialize method

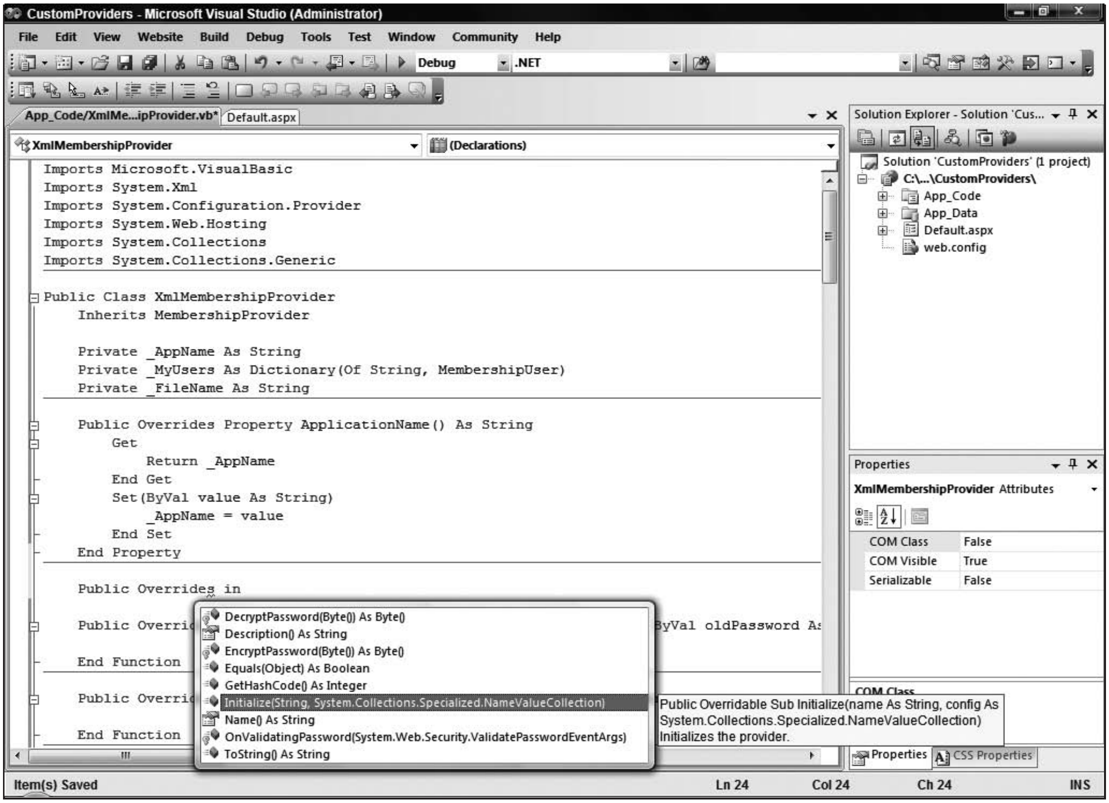
VB

```
Public Overrides Sub Initialize(ByVal name As String, _
   ByVal config As System.Collections.Specialized.NameValueCollection)
```

```
        MyBase.Initialize(name, config)
End Sub
```

C#

```
public override void Initialize(string name,
  System.Collections.Specialized.NameValueCollection config)
{
   base.Initialize(name, config);
}
```



CustomProviders - Microsoft Visual Studio (Administrator)
File
Edit
View
Website
Build
Debug
Tools
Test
Window
Community
Help
Debug
.NET
App_Code/XmlMe...ipProvider.vb*
Default.aspx
XmlMembershipProvider
(Declarations)
Solution Explorer - Solution 'Cus...
Solution 'CustomProviders' (1 project)
C:\...\CustomProviders\
App_Code
App_Data
Default.aspx
web.config
Properties
XmlMembershipProvider Attributes
COM Class
Properties
CSS Properties
Item(s) Saved
Ln 24
Col 24
Ch 24
INS

```
Imports Microsoft.VisualBasic
Imports System.Xml
Imports System.Configuration.Provider
Imports System.Web.Hosting
Imports System.Collections
Imports System.Collections.Generic

Public Class XmlMembershipProvider
    Inherits MembershipProvider

    Private _AppName As String
    Private _MyUsers As Dictionary(Of String, MembershipUser)
    Private _FileName As String

    Public Overrides Property ApplicationName() As String
        Get
            Return _AppName
        End Get
        Set(ByVal value As String)
            _AppName = value
        End Set
    End Property

    Public Overrides in

    Public Overri                                          ByVal oldPassword As

    End Function

    Public Overri

    End Function
```

DecryptPassword(Byte()) As Byte()
Description() As String
EncryptPassword(Byte()) As Byte()
Equals(Object) As Boolean
GetHashCode() As Integer
Initialize(String, System.Collections.Specialized.NameValueCollection)
Name() As String
OnValidatingPassword(System.Web.Security.ValidatePasswordEventArgs)
ToString() As String

COM Class	False
COM Visible	True
Serializable	False

Public Overridable Sub Initialize(name As String, config As System.Collections.Specialized.NameValueCollection)
Initializes the provider.

Figure 13-5

The `Initialize()` method takes two parameters. The first parameter is the name of the parameter. The second is the name/value collection from the provider declaration in the `web.config` file. This includes all the attributes and their values, such as the `xmlUserDatabaseFile` attribute and the value of the name of the XML file that holds the user information. Using `config`, you can gain access to these defined values.

For the `XmlFileProvider` instance, you address the `applicationName` attribute and the `xmlUserDatabaseFile` attribute. You do this as shown in Listing 13-14.

Listing 13-14: Extending the Initialize() method

VB

```
Public Overrides Sub Initialize(ByVal name As String, _
  ByVal config As System.Collections.Specialized.NameValueCollection)
        MyBase.Initialize(name, config)

        _AppName = config("applicationName")

        If (String.IsNullOrEmpty(_AppName)) Then
            _AppName = "/"
        End If

        _FileName = config("xmlUserDatabaseFile")

        If (String.IsNullOrEmpty(_FileName)) Then
            _FileName = "~/App_Data/Users.xml"
        End If
End Sub
```

C#

```
public override void Initialize(string name,
  System.Collections.Specialized.NameValueCollection config)
{
   base.Initialize(name, config);

   _AppName = config["applicationName"];

   if (String.IsNullOrEmpty(_AppName))
   {
      _AppName = "/";
   }

   _FileName = config["xmlUserDatabaseFile"];

   if (String.IsNullOrEmpty(_FileName))
   {
      _FileName = "~/App_Data/Users.xml";
   }
}
```

Besides performing the initialization using `MyBase.Initialize()`, you retrieve both the `applicationName` and `xmlUserDatabaseFile` attribute's values using `config`. In all cases, you should first check whether the value is either null or empty. You use the `String.IsNullOrEmpty()` method to assign default values if the attribute is missing for the provider declaration in the `web.config` file. In the case of the `XmlFileProvider` instance, this is, in fact, the case. The `applicationName` attribute in the `XmlFileProvider` declaration is actually not declared and, for this reason, the default value of `/` is actually assigned as the value.

In the case of the `xmlUserDatabaseFile` attribute, a value is provided. If no value is provided in the `web.config` file, the provider looks for an XML file named `Users.xml` found in the App_Data folder.

Validating Users

One of the more important features of the membership provider is that it validates users (it authenticates them). The validation of users is accomplished through the ASP.NET Login server control. This control, in turn, makes use of the `Membership.ValidateUser()` method that ends up using the `ValidateUser()` method in the `XmlMembershipProvider` class.

Now that the `Initialize()` method and private variables are in place, you can start giving the provider some functionality. The implementation of the `ValidateUser()` method is presented in Listing 13-15.

Listing 13-15: Implementing the ValidateUser() method

VB

```
Public Overrides Function ValidateUser(ByVal username As String, _
  ByVal password As String) As Boolean

   If (String.IsNullOrEmpty(username) Or String.IsNullOrEmpty(password)) Then
      Return False
   End If

   Try
      ReadUserFile()
      Dim mu As MembershipUser

      If (_MyUsers.TryGetValue(username.ToLower(), mu)) Then
         If (mu.Comment = password) Then
            Return True
         End If
      End If

      Return False
   Catch ex As Exception
      Throw New Exception(ex.Message.ToString())
   End Try
End Function
```

C#

```
public override bool ValidateUser(string username, string password)
{
   if (String.IsNullOrEmpty(username) || String.IsNullOrEmpty(password))
   {
      return false;
   }

   try
   {
      ReadUserFile();

      MembershipUser mu;

      if (_MyUsers.TryGetValue(username.ToLower(), out mu))
```

Continued

```
            {
               if (mu.Comment == password)
               {
                  return true;
               }
            }

            return false;
         }
         catch (Exception ex)
         {
            throw new Exception(ex.Message.ToString());
         }
      }
```

Looking over the `ValidateUser()` method, you can see that it takes two parameters, the username and the password of the user (both of type `String`). The value returned from `ValidateUser()` is a `Boolean` — just a `True` or `False` value to inform of the success for failure of the validation process.

One of the first operations performed in the `ValidateUser()` method is a check to determine whether either the username or the password is missing from the invocation. If one of these items is missing in the request, a `False` value is returned.

From there, a `Try Catch` is done to check if the user and the user's password are included in the XML file. The process of getting the user information out of the XML file and into the `MyUsers` variable is done by the `ReadUserFile()` method. This method is described shortly, but the important concept is that the `_MyUsers` variable is an instance of the `Dictionary` generic class. The key is a lowercase string value of the username, whereas the value is of type `MembershipUser`, a type provided via the membership system.

After the `_MyUsers` object is populated with all users in the XML file, a `MembershipUser` instance is created. This object is the output of a `TryGetValue` operation. The `MembershipUser` does not contain the password of the user and, for this reason, the `ReadUserFile()` method makes the user's password the value of the `Comment` property of the `MembershipUser` class. If the username is found in the dictionary collection, then the password of that particular `MembershipUser` instance is compared to the value in the `Comment` property. The return value from the `ValidateUser()` method is `True` if they are found to be the same.

As you can see, this method really is dependent upon the results which come from the `ReadUserFile()` method, which is covered next.

Building the ReadUserFile() Method

The `ReadUserFile()` method reads the contents of the XML file that contains all the users for the application. This method is a custom method, and its work is done outside of the `ValidateUser()` method. This means it can be reused in other methods you might want to implement (such as the `GetAllUsers()` method). The only job of the `ReadUserFile()` method is to read the contents of the XML file and place all the users in the `_MyUsers` variable, as illustrated in Listing 13-16.

Listing 13-16: The ReadUserFile() method to get all the users of the application

VB

```
Private Sub ReadUserFile()
   If (_MyUsers Is Nothing) Then
      SyncLock (Me)
         _MyUsers = New Dictionary(Of String, MembershipUser)()
         Dim xd As XmlDocument = New XmlDocument()
         xd.Load(HostingEnvironment.MapPath(_FileName))
         Dim xnl As XmlNodeList = xd.GetElementsByTagName("User")

         For Each node As XmlNode In xnl
            Dim mu As MembershipUser = New MembershipUser(Name, _
               node("Username").InnerText, _
               Nothing, _
               node("Email").InnerText, _
               String.Empty, _
               node("Password").InnerText, _
               True, _
               False, _
               DateTime.Parse(node("DateCreated").InnerText), _
               DateTime.Now, _
               DateTime.Now, _
               DateTime.Now, _
               DateTime.Now)

               _MyUsers.Add(mu.UserName.ToLower(), mu)
         Next
      End SyncLock
   End If
End Sub
```

C#

```
private void ReadUserFile()
{
      if (_MyUsers == null)
      {
        lock (this)
        {
         _MyUsers = new Dictionary<string, MembershipUser>();
         XmlDocument xd = new XmlDocument();
         xd.Load(HostingEnvironment.MapPath(_FileName));
         XmlNodeList xnl = xd.GetElementsByTagName("User");

         foreach (XmlNode node in xnl)
         {
            MembershipUser mu = new MembershipUser(Name,
               node["Username"].InnerText,
               null,
               node["Email"].InnerText,
               String.Empty,
               node["Password"].InnerText,
               true,
```

Continued

```
                    false,
                    DateTime.Parse(node["DateCreated"].InnerText),
                    DateTime.Now,
                    DateTime.Now,
                    DateTime.Now,
                    DateTime.Now);

                    _MyUsers.Add(mu.UserName.ToLower(), mu);
                }
            }
        }
    }
```

The first action of the `ReadUserFile()` method is to place a lock on the action that is going to occur in the thread being run. This is a unique feature in ASP.NET. When you are writing your own providers, be sure you use thread-safe code. For most items that you write in ASP.NET, such as an HttpModule or an HttpHandler (covered in Chapter 27), you don't need to make them thread-safe. These items may have multiple requests running on multiple threads, and each thread making a request to either the HttpModule or the HttpHandler sees a unique instance of these items.

Unlike an HttpHandler, only one instance of a provider is created and utilized by your ASP.NET application. If there are multiple requests being made to your application, all these threads are trying to gain access to the single provider instance contained in the application. Because more than one request might be coming into the provider instance at the same time, you should create the provider in a thread-safe manner. This can be accomplished by using a lock operation when performing tasks such as a file I/O operation. This is the reason for the use of the `SyncLock` (for Visual Basic) and the `lock` (for C#) statements in the `ReadUserFile()` method.

The advantage to all of this, however, is that a single instance of the provider is running in your application. After the `_MyUsers` object is populated with the contents of the XML file, you have no need to repopulate the object. The provider instance doesn't just disappear after a response is issued to the requestor. Instead, the provider instance is contained in memory and utilized for multiple requests. This is the reason for checking whether `_MyUsers` contains any values before reading the XML file.

If you find that `_MyUsers` is null, use the `XmlDocument` object to get at every `<User>` element in the document. For each `<User>` element in the document, the values are assigned to a `MembershipUser` instance. The `MembershipUser` object takes the following arguments:

```
MembershipUser(
   providerName As String, _
   name As String, _
   providerUserKey As Object, _
   email As String, _
   passwordQuestion As String, _
   comment As String, _
   isApproved As Boolean, _
   isLockedOut As Boolean, _
   creationDate As DateTime, _
   lastLoginDate As DateTime, _
   lastActivityDate As DateTime, _
   lastPasswordChangedDate As DateTime, _
   lastLockoutDate As DateTime)
```

Although you do not provide a value for each and every item in this construction, the values that are really needed are pulled from the XML file using the `XmlNode` object. Then after the `MembershipUser` object is populated with everything you want, the next job is to add this to the `_MyUsers` object using the following:

```
_MyUsers.Add(mu.UserName.ToLower(), mu)
```

With the `ReadUserFile()` method in place, as stated, you can now use this in more than the `ValidateUser()` method. Remember that once the `_MyUsers` collection is populated, you don't need to repopulate the collection again. Instead, it remains in place for the other methods to make use of. Next, this chapter looks at using what has been demonstrated so far in your ASP.NET application.

Using the XmlMembershipProvider for User Login

If you have made it this far in the example, you do not need to do much more to make use of the `XmlMembershipProvider` class. At this point, you should have the XML data file in place that is a representation of all the users of your application (this XML file was presented in Listing 13-7) as well as the `XmlFileProvider` declaration in the `web.config` file of your application (the changes to the `web.config` file are presented in Listing 13-8). Of course, another necessary item is either the `XmlMembershipProvider.vb` or `.cs` class in the `App_Code` folder of your application. However, if you built the provider as a class library, you want to just make sure the DLL created is referenced correctly in your ASP.NET application (which means the DLL is in the Bin folder). After you have these items in place, it is pretty simple to start using the provider.

For a quick example of this, simply create a `Default.aspx` page that has only the text: `You are authenticated!`

Next, you create a `Login.aspx` page, and you place a single Login server control on the page. You won't need to make any other changes to the `Login.aspx` page besides these. Users can now log in to the application.

For information on the membership system, which includes detailed explanations of the various server controls it offers, visit Chapter 16.

When you have those two files in place within your mini-ASP.NET application, the next step is to make some minor changes to the `web.config` to allow for Forms authentication and to deny all anonymous users to view any of the pages. This bit of code is presented in Listing 13-17.

Listing 13-17: Denying anonymous users to view the application in the web.config file

```
<configuration>
   <system.web>

      <authentication mode="Forms"/>
      <authorization>
         <deny users="?"/>
      </authorization>

      <!-- Other settings removed for clarity -->

   </system.web>
</configuration>
```

Now, run the `Default.aspx` page, and you are immediately directed to the `Login.aspx` page (you should have this file created in your application and it should contain only a single Login server control) where you apply one of the username and password combinations that are present in the XML file. It is as simple as that!

The nice thing with the provider-based model found in ASP.NET 3.5 is that the controls that are working with the providers don't know the difference when these large changes to the underlying provider are made. In this example, you have removed the default `SqlMembershipProvider` and replaced it with a read-only XML provider, and the Login server control is really none the wiser. When the end user clicks the Log In button within the Login server control, the control is still simply making use of the `Membership.ValidateUser()` method, which is working with the `XmlMembershipProvider` that was just created. As you should see by now, this is a powerful model.

Extending Pre-Existing Providers

In addition to building your own providers from one of the base abstract classes such as `MembershipProvider`, another option is to simply extend one of the pre-existing providers that come with ASP.NET.

For instance, you might be interested in using the membership and role management systems with SQL Server but want to change how the default providers (`SqlMembershipProvider` or `SqlRoleProvider`) work under the covers. If you are going to work with an underlying data store that is already utilized by one of the providers available out of the box, then it actually makes a lot more sense to change the behavior of the available provider rather than build a brand-new provider from the ground up.

The other advantage of working from a pre-existing provider is that there is no need to override everything the provider exposes. Instead, if you are interested in changing only a particular behavior of a built-in provider, you might only need to override a couple of the exposed methods and nothing more, making this approach rather simple and quick to achieve in your application.

Next, this chapter looks at extending one of the built-in providers to change the underlying functionality of the provider.

Limiting Role Capabilities with a New LimitedSqlRoleProvider Provider

Suppose you want to utilize the new role management system in your ASP.NET application and have every intention of using a SQL Server backend for the system. Suppose you also want to limit what roles developers can create in their applications, and you want to remove their capability to add users to a particular role in the system.

Instead of building a role provider from scratch from the `RoleProvider` abstract class, it makes more sense to derive your provider from `SqlRoleProvider` and to simply change the behavior of a few methods that deal with the creation of roles and adding users to roles.

For this example, create the provider in your application within the `App_Code` folder as before. In reality, however, you probably want to create a Class Library project if you want to use this provider across your company so that your development teams can use a DLL rather than a modifiable class file.

Within the `App_Code` folder, create a class file called `LimitedSqlRoleProvider.vb` or `.cs`. You want this class to inherit from `SqlRoleProvider`, and this gives you the structure shown in Listing 13-18.

Listing 13-18: The beginnings of the LimitedSqlRoleProvider class

VB

```
Imports Microsoft.VisualBasic
Imports System.Configuration.Provider
Public Class LimitedSqlRoleProvider
    Inherits SqlRoleProvider

End Class
```

C#

```
using System;
using System.Web;
using System.Web.Security;
using System.Configuration;
using System.Configuration.Provider;

public class LimitedSqlRoleProvider : SqlRoleProvider
{

}
```

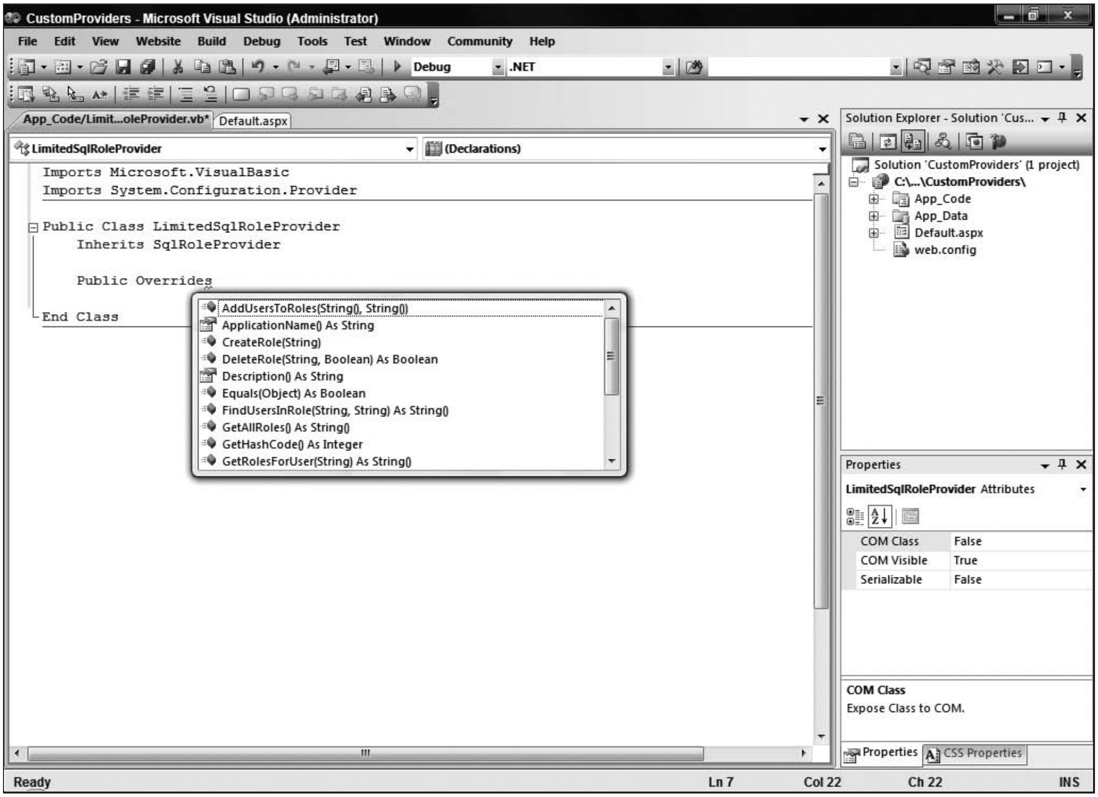
This is similar to creating the `XmlMembershipProvider` class. When you did that, however, you were able to use Visual Studio to build the entire class skeleton of all the methods and properties you had to override to get the new class up and running. In this case, if you try to do the same thing in Visual Studio, you get an error (if using C#) or, perhaps, no result at all (if using Visual Basic) because you are not working with an abstract class. You do not need to override an enormous number of methods and properties. Instead, because you are deriving from a class that already inherits from one of these abstract classes, you can get by with overriding only the methods and properties that you need to work with and nothing more.

To get at this list of methods and properties within Visual Studio, you simply type **Public Overrides** (when using Visual Basic) or **public override** (when using C#). IntelliSense then provides you with a large drop-down list of available methods and properties to work with, as illustrated in Figure 13-6.

For this example, you only override the `CreateRole()`, `AddUsersToRoles()`, and `DeleteRole()` methods. These are described next.

The CreateRole() Method

The `CreateRole()` method in the `SqlRoleProvider` class allows developers to add any role to the system. The only parameter required for this method is a string value that is the name of the role. For this example, instead of letting developers create any role they wish, this provider limits the role creation to only the *Administrator* and *Manager* roles. To accomplish this in the `CreateRole()` method, you code the method as presented in Listing 13-19.


CustomProviders - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Tools Test Window Community Help
Debug
.NET
App_Code/Limit...oleProvider.vb*
Default.aspx
LimitedSqlRoleProvider
(Declarations)
AddUsersToRoles(String(), String())
ApplicationName() As String
CreateRole(String)
DeleteRole(String, Boolean) As Boolean
Description() As String
Equals(Object) As Boolean
FindUsersInRole(String, String) As String()
GetAllRoles() As String()
GetHashCode() As Integer
GetRolesForUser(String) As String()
Solution Explorer - Solution 'Cus...
Solution 'CustomProviders' (1 project)
C:\...\CustomProviders\
App_Code
App_Data
Default.aspx
web.config
Properties
LimitedSqlRoleProvider Attributes
COM Class
Expose Class to COM.
Properties
CSS Properties
Ready
Ln 7
Col 22
Ch 22
INS

```
Imports Microsoft.VisualBasic
Imports System.Configuration.Provider

Public Class LimitedSqlRoleProvider
    Inherits SqlRoleProvider

    Public Overrides

End Class
```

COM Class False
COM Visible True
Serializable False

Figure 13-6

Listing 13-19: Allowing only the Administrator or Manager role in the CreateUser() method

VB

```
Public Overrides Sub CreateRole(ByVal roleName As String)
   If (roleName = "Administrator" Or roleName = "Manager") Then
     MyBase.CreateRole(roleName)
   Else
     Throw New _
      ProviderException("Role creation limited to only Administrator and Manager")
   End If
End Sub
```

C#

```
public override void CreateRole(string roleName)
{
   if (roleName == "Administrator" || roleName == "Manager")
   {
      base.CreateRole(roleName);
   }
```

```
      else
      {
        throw new
          ProviderException("Role creation limited to only Administrator and Manager");
      }
   }
```

In this method, you can see that a check is first done to determine whether the role being created is either Administrator or Manager. If the role being created is not one of these defined roles, a `ProviderException` is thrown informing the developer of which roles they are allowed to create.

If Administrator or Manager is one of the roles, then the base class (`SqlRoleProvider`) `CreateRole()` method is invoked.

The DeleteRole() Method

If you allow developers using this provider to create only specific roles, you might not want them to delete any role after it is created. If this is the case, you want to override the `DeleteRole()` method of the `SqlRoleProvider` class, as illustrated in Listing 13-20.

Listing 13-20: Disallowing the DeleteRole() method

VB

```
Public Overrides Function DeleteRole(ByVal roleName As String, _
  ByVal throwOnPopulatedRole As Boolean) As Boolean
        Return False
End Function
```

C#

```
public override bool DeleteRole(string roleName, bool throwOnPopulatedRole)
{
   return false;
}
```

Looking at the `DeleteRole()` method, you can see that deleting any role is completely disallowed. Instead of raising the base class's `DeleteRole()` and returning the following:

```
Return MyBase.DeleteRole(roleName, throwOnPopulatedRole)
```

a `False` value is returned and no action is taken. Another approach is to throw a `NotSupportedException`, as shown here:

```
Throw New NotSupportedException()
```

The AddUsersToRoles() Method

As you look over the methods that can be overridden, notice that only one single method allows you to add any number of users to any number of roles. Multiple methods in the `Roles` class actually map to this method. If you look at the `Roles` class, notice the `AddUserToRole()`, `AddUserToRoles()`, `AddUsersToRole()`, and `AddUsersToRoles()` methods at your disposal. All these actually map to the `AddUsersToRoles()` method that is available in the `RoleProvider` base class.

Suppose you want, for example, to enable developers to add users only to the Manager role but not to add any users to the Administrator role. You could accomplish something like this by constructing a method, as shown in Listing 13-21.

Listing 13-21: Disallowing users to be added to a particular role

VB

```
Public Overrides Sub AddUsersToRoles(ByVal usernames() As String, _
  ByVal roleNames() As String)

   For Each roleItem As String In roleNames
      If roleItem = "Administrator" Then
         Throw New _
            ProviderException("You are not authorized to add any users" & _
               " to the Administrator role")
      End If
   Next

   MyBase.AddUsersToRoles(usernames, roleNames)
End Sub
```

C#

```
public override void AddUsersToRoles(string[] usernames, string[] roleNames)
{
   foreach (string roleItem in roleNames)
   {
      if (roleItem == "Administrator")
      {
         throw new ProviderException("You are not authorized to add any users" +
           " to the Administrator role");
      }
   }

   base.AddUsersToRoles(usernames, roleNames);
}
```

This overridden method iterates through all the provided roles, and if one of the roles contained in the string array is the role Administrator, then a `ProviderException` instance is thrown informing the developer that he or she is not allowed to add any users to this particular role. Although it is not shown here, you can also take the same approach with the `RemoveUsersFromRoles()` method exposed from the `RoleProvider` base class.

Using the New LimitedSqlRoleProvider Provider

After you have the provider in place and ready to use, you have to make some modifications to the `web.config` file in order to use this provider in your ASP.NET application. You learn how you add what you need to the `web.config` file for this provider in Listing 13-22.

Listing 13-22: Making the appropriate changes to the web.config file for the provider

```
<configuration>
   <system.web>
```

```
    <roleManager defaultProvider="LimitedProvider" enabled="true">
      <providers>
        <add connectionStringName="LocalSqlServer" applicationName="/"
         name="LimitedProvider"
         type="LimitedSqlRoleProvider" />
      </providers>
    </roleManager>

   </system.web>
</configuration>
```

Remember that you have to define the provider to use in your application by providing a value for the `defaultProvider` attribute and defining that provider further in the `<provider>` section. You also have to enable the provider by setting the `enabled` attribute to `true`. By default, the role management system is disabled.

Using the `<add>` element, you can add a provider instance that makes use of the `LimitedSqlRoleProvider` class. Because this provider derives from the `SqlRoleProvider` class, you must use some of the same attributes that this provider requires, such as the `connectionStringName` attribute that points to the connection string to use to connect to the specified SQL instance.

After you have the new `LimitedSqlRoleProvider` instance in place and defined in the `web.config` file, you can use the `Roles` class in your application just as you normally would, but notice the behavior of this class is rather different from the normal `SqlRoleProvider`.

To see it in action, construct a simple ASP.NET page that includes a TextBox, Button, and Label server control. The page should appear as shown in Listing 13-23.

Listing 13-23: Using Roles.CreateRole()

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)

        Try
            Roles.CreateRole(TextBox1.Text)
            Label1.Text = "Role successfully created."
        Catch ex As Exception
            Label1.Text = ex.Message.ToString()
        End Try
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Main Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
```

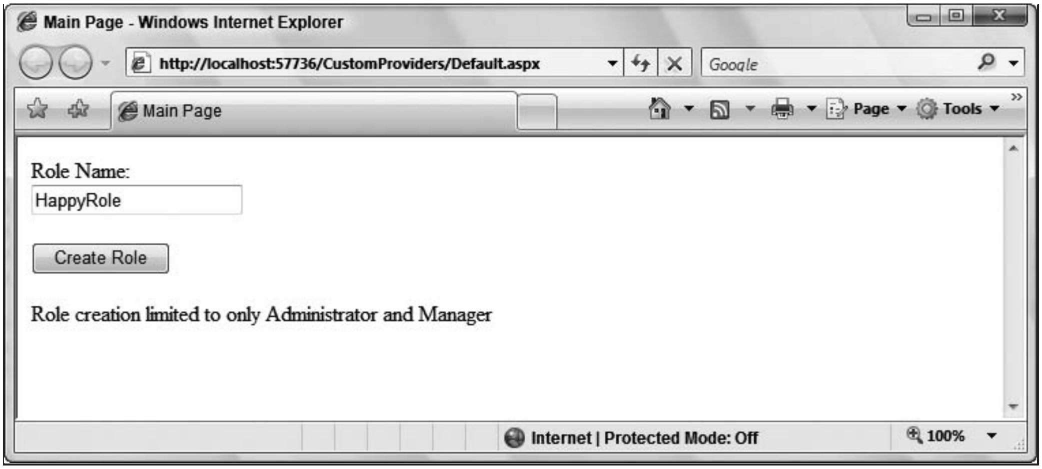
Continued

```
            Role Name:<br />
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
            <br />
            <asp:Button ID="Button1" runat="server" Text="Create Role"
             OnClick="Button1_Click" /><br />
            <br />
            <asp:Label ID="Label1" runat="server"></asp:Label></div>
        </form>
    </body>
    </html>
```

C#

```
    <%@ Page Language="C#" %>

    <script runat="server">
        protected void Button1_Click(object sender, EventArgs e)
        {
            try
            {
                Roles.CreateRole(TextBox1.Text);
                Label1.Text = "Role successfully created.";
            }
            catch (Exception ex)
            {
                Label1.Text = ex.Message.ToString();
            }
        }
    </script>
```

This simple ASP.NET page enables you to type in a string value in the text box and to attempt to create a new role using this value. Note that anything other than the role Administrator and Manager results in an error. So, when the `Roles.CreateRole()` is called, an error is produced if the rules defined by the provider are not followed. In fact, running this page and typing in a role other than the Administrator or Manager role gives you the results presented in Figure 13-7.


Main Page - Windows Internet Explorer
http://localhost:57736/CustomProviders/Default.aspx
Google
Main Page
Page
Tools
Internet | Protected Mode: Off
100%

Role Name:

HappyRole

Create Role

Role creation limited to only Administrator and Manager

Figure 13-7

To show this provider in action, create another ASP.NET page that allows you to add users to a particular role. As stated, this can be done with a number of available methods, but in this case, this example uses the `Roles.AddUserToRole()` method. This is illustrated in Listing 13-24.

Listing 13-24: Attempting to add users to a role through the new role provider

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)

        Try
            Roles.AddUserToRole(TextBox1.Text, TextBox2.Text)
            Label1.Text = "User successfully added to role"
        Catch ex As Exception
            Label1.Text = ex.Message.ToString()
        End Try
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Main Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        Add the following user:<br />
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
        <br />
        To role:<br />
        <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox><br />
        <br />
        <asp:Button ID="Button1" runat="server" Text="Add User to Role"
         OnClick="Button1_Click" /><br />
        <br />
        <asp:Label ID="Label1" runat="server"></asp:Label></div>
    </form>
</body>
</html>
```

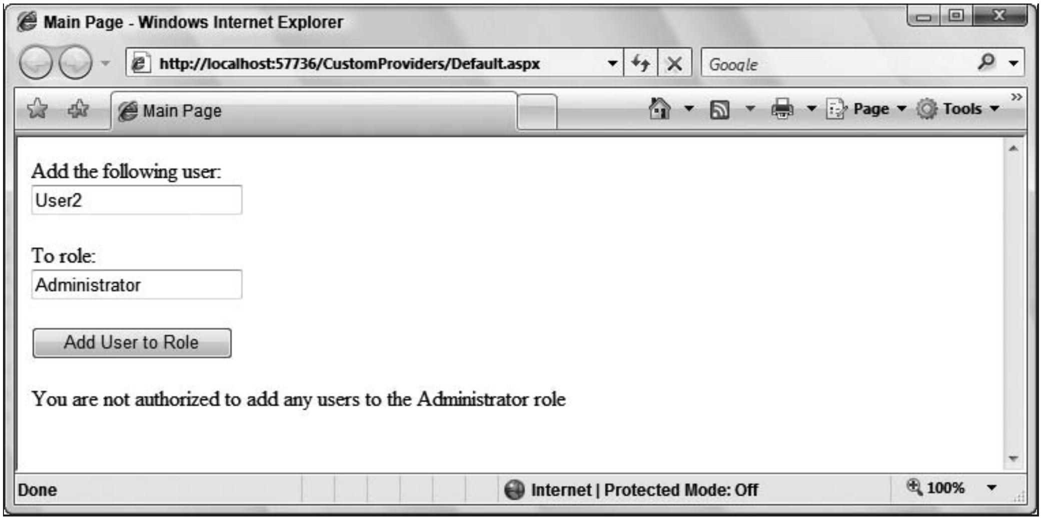
C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        try
        {
            Roles.AddUserToRole(TextBox1.Text, TextBox2.Text);
```

Continued

```
                Label1.Text = "User successfully added to role";
            }
            catch (Exception ex)
            {
                Label1.Text = ex.Message.ToString();
            }
        }
    </script>
```

In this example, two text boxes are provided. The first asks for the username and the second asks for the role to add the user to. The code for the button click event uses the `Roles.AddUserToRole()` method. Because you built the provider, you know that an error is thrown if there is an attempt to add a user to the Administrator role. This attempt is illustrated in Figure 13-8.


Main Page - Windows Internet Explorer
http://localhost:57736/CustomProviders/Default.aspx
Google
Main Page
Page
Tools
Add the following user:
User2
To role:
Administrator
Add User to Role
Done
Internet | Protected Mode: Off
100%

You are not authorized to add any users to the Administrator role

Figure 13-8

In this case, there was an attempt to add User2 to the Administrator role. This, of course, throws an error and returns the error message that is defined in the provider.

Summary

From this chapter and the last chapter, you got a good taste of the provider model and what it means to the ASP.NET 3.5 applications you build today. Although a lot of providers are available to you out of the box to use for interacting with one of the many systems provided in ASP.NET, you are not limited to just these providers. You definitely can either build your own providers or even extend the functionality of the providers already present in the system.

This chapter looked at both of these scenarios. First, you built your own provider to use the membership system with an XML data store for the user data, and then you worked through an example of extending the `SqlRoleProvider` class (something already present in ASP.NET) to change the underlying behavior of this provider.

14

Site Navigation

The Web applications that you develop generally have more than a single page to them. Usually you create a number of Web pages that are interconnected in some fashion. If you also build the navigation around your collection of pages, you make it easy for the end user to successfully work through your application in a straightforward manner.

Currently, you must choose among a number of different ways to expose the paths through your application to the end user. The difficult task of site navigation is compounded when you continue to add pages to the overall application.

The present method for building navigation within Web applications is to sprinkle pages with hyperlinks. Hyperlinks are generally added to Web pages by using include files or user controls. They can also be directly hard-coded onto a page so that they appear in the header or the sidebar of the page being viewed. The difficulties in working with navigation become worse when you move pages around or change page names. Sometimes developers are forced to go to each and every page in the application just to change some aspect of the navigation.

ASP.NET 3.5 tackles this problem by providing a navigation system that makes it quite trivial to manage how end users work through the applications you create. This capability in ASP.NET is complex; but the great thing is that it can be as simple as you need it to be, or you can actually get in deep and control every aspect of how it works.

The site navigation system includes the capability to define your entire site in an XML file that is called a *site map*. After you define a site map, you can work with it programmatically using the `SiteMap` class. Another aspect of the sitemap capability available in ASP.NET is a data provider that is specifically developed to work with site map files and to bind them to a series of navigation-based server controls. This chapter looks at all these components in the ASP.NET 3.5 navigation system. The following section introduces site maps.

XML-Based Site Maps

Although a site map is not a required element (as you see later), one of the common first steps you take in working with the ASP.NET 3.5 navigation system is building a site map for your application. A site map is an XML description of your site's structure.

You use this site map to define the navigational structure of all the pages in your application and how they relate to one another. If you do this according to the ASP.NET site map standard, you can then interact with this navigation information using either the `SiteMap` class or the SiteMapDataSource control. By using the SiteMapDataSource control, you can then bind the information in the site map file to a variety of data-binding controls, including the navigation server controls provided by ASP.NET.

To create a site map file for your application, add a site map or an XML file to your application. When asked, you name the XML file `Web.sitemap`; this file is already in place if you select the Site Map option. The file is named `Web` and has the file extension of `.sitemap`. Take a look at an example of a `.sitemap` file illustrated here in Listing 14-1.

Listing 14-1: An example of a Web.sitemap file

```
<?xml version="1.0" encoding="utf-8" ?>

<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
   <siteMapNode title="Home" description="Home Page" url="Default.aspx">
      <siteMapNode title="News" description="The Latest News" url="News.aspx">
         <siteMapNode title="U.S." description="U.S. News"
          url="News.aspx?cat=us" />
         <siteMapNode title="World" description="World News"
          url="News.aspx?cat=world" />
         <siteMapNode title="Technology" description="Technology News"
          url="News.aspx?cat=tech" />
         <siteMapNode title="Sports" description="Sports News"
          url="News.aspx?cat=sport" />
      </siteMapNode>
      <siteMapNode title="Finance" description="The Latest Financial Information"
        url="Finance.aspx">
         <siteMapNode title="Quotes" description="Get the Latest Quotes"
          url="Quotes.aspx" />
         <siteMapNode title="Markets" description="The Latest Market Information"
          url="Markets.aspx">
            <siteMapNode title="U.S. Market Report"
             description="Looking at the U.S. Market" url="MarketsUS.aspx" />
            <siteMapNode title="NYSE"
             description="The New York Stock Exchange" url="NYSE.aspx" />
         </siteMapNode>
         <siteMapNode title="Funds" description="Mutual Funds"
          url="Funds.aspx" />
      </siteMapNode>
      <siteMapNode title="Weather" description="The Latest Weather"
       url="Weather.aspx" />
   </siteMapNode>
</siteMap>
```

So what does this file give you? Well, it gives you a logical structure that ASP.NET can now use in the rest of the navigation system it provides. Next, this chapter examines how this file is constructed.

The root node of this XML file is a `<siteMap>` element. Only one `<siteMap>` element can exist in the file. Within the `<siteMap>` element, there is a single root `<siteMapNode>` element. This is generally the start page of the application. In the case of the file in Listing 14-1, the root `<siteMapNode>` points to the `Default.aspx` page, the start page:

```
<siteMapNode title="Home" description="Home Page" url="Default.aspx">
```

The following table describes the most common attributes in the `<siteMapNode>` element.

Attribute	Description
`title`	The `title` attribute provides a textual description of the link. The `String` value used here is the text used for the link.
`description`	The `description` attribute not only reminds you what the link is for, but it is also used for the `ToolTip` attribute on the link. The `ToolTip` attribute is the yellow box that shows up next to the link when the end user hovers the cursor over the link for a couple of seconds.
`url`	The `url` attribute describes where the file is located in the solution. If the file is in the root directory, simply use the file name, such as `"Default.aspx"`. If the file is located in a subfolder, be sure to include the folders in the String value used in this attribute. For example, ‘‘`MySubFolder/Markets.aspx`’’.

After you have the first `<siteMapNode>` in place, you can then nest as many additional `<siteMapNode>` elements as you need within the root `<siteMapNode>` element. You can also create additional link-levels by creating child `<siteMapNode>` elements for any parent `<siteMapNode>` in the structure.

The example in Listing 14-1 gives the application the following navigational structure:

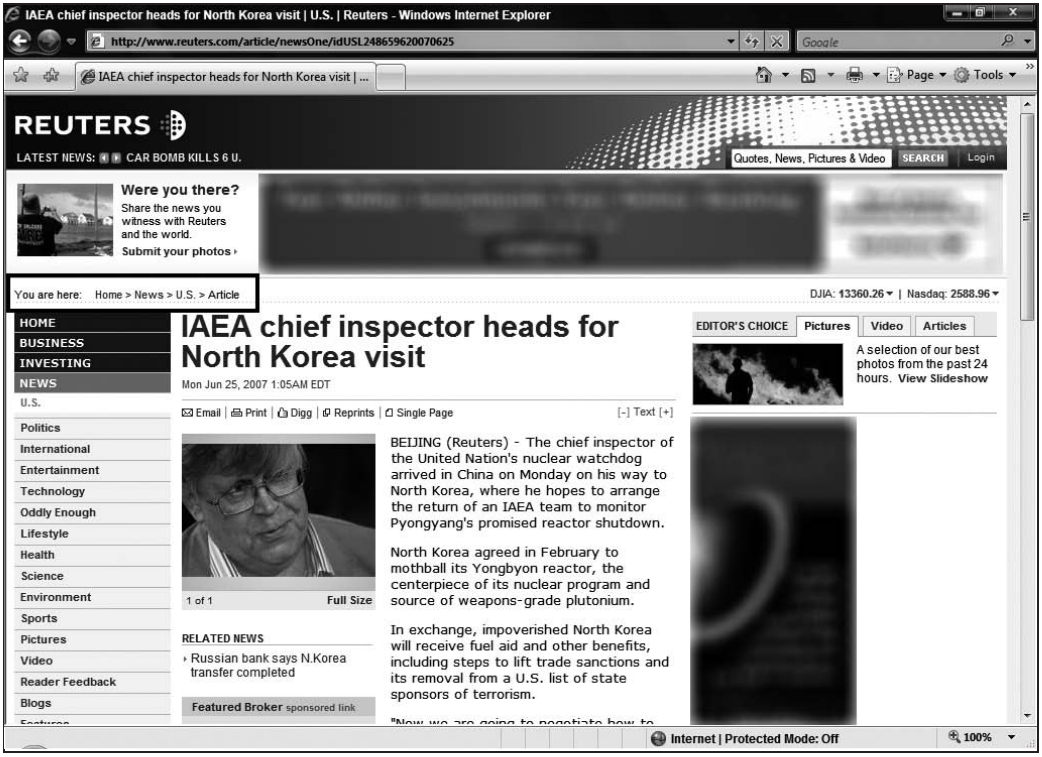
```
Home
   News
      U.S.
      World
      Technology
      Sports
   Finance
      Quotes
      Markets
         U.S. Market Report
         NYSE
      Funds
   Weather
```

You can see that this structure goes down three levels in some places. One of the easiest places to use this file is with the SiteMapPath server control that comes with ASP.NET. The SiteMapPath server control in ASP.NET is built to work specifically with the `.sitemap` files.

SiteMapPath Server Control

It is quite easy to use the `.sitemap` file you just created with the SiteMapPath server control provided with ASP.NET. You can find this control in the Navigation section of the Visual Studio 2008 IDE.

The SiteMapPath control creates navigation functionality that you once might have either created yourself or have seen elsewhere in Web pages on the Internet. The SiteMapPath control creates what some refer to as *breadcrumb navigation*. This is a linear path defining where the end user is in the navigation structure. The `Reuters.com` Web site, shown in Figure 14-1, uses this type of navigation. A black box shows the breadcrumb navigation used on the page.


IAEA chief inspector heads for North Korea visit | U.S. | Reuters - Windows Internet Explorer
http://www.reuters.com/article/newsOne/idUSL248659620070625
Google
IAEA chief inspector heads for North Korea visit | ...
Page
Tools
REUTERS
LATEST NEWS: CAR BOMB KILLS 6 U.
Quotes, News, Pictures & Video
SEARCH
Login
Were you there?
Share the news you witness with Reuters and the world.
Submit your photos
You are here: Home > News > U.S. > Article
DJIA: 13360.26 | Nasdaq: 2588.96
HOME
BUSINESS
INVESTING
NEWS
U.S.
Politics
International
Entertainment
Technology
Oddly Enough
Lifestyle
Health
Science
Environment
Sports
Pictures
Video
Reader Feedback
Blogs
IAEA chief inspector heads for North Korea visit
Mon Jun 25, 2007 1:05AM EDT
Email | Print | Digg | Reprints | Single Page
[-] Text [+]
1 of 1
Full Size
RELATED NEWS
Russian bank says N.Korea transfer completed
Featured Broker sponsored link
EDITOR'S CHOICE
Pictures
Video
Articles
A selection of our best photos from the past 24 hours. View Slideshow
Internet | Protected Mode: Off
100%

BEIJING (Reuters) - The chief inspector of the United Nation's nuclear watchdog arrived in China on Monday on his way to North Korea, where he hopes to arrange the return of an IAEA team to monitor Pyongyang's promised reactor shutdown.

North Korea agreed in February to mothball its Yongbyon reactor, the centerpiece of its nuclear program and source of weapons-grade plutonium.

In exchange, impoverished North Korea will receive fuel aid and other benefits, including steps to lift trade sanctions and its removal from a U.S. list of state sponsors of terrorism.

"Now we are going to negotiate how to

Figure 14-1

The purpose of this type of navigation is to show end users where they are in relation to the rest of the site. Traditionally, coding this kind of navigation has been tricky, to say the least; but now with the introduction of the SiteMapPath server control, you should find coding for this type of navigation a breeze.

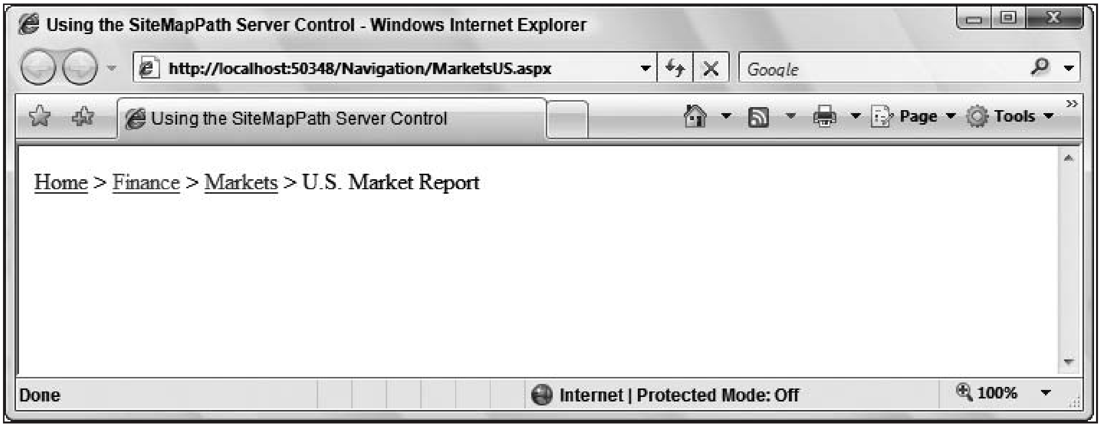
You should first create an application that has the `Web.sitemap` file created in Listing 14-1. From there, create a WebForm called `MarketsUS.aspx`. This file is defined in the `Web.sitemap` file as being on the lowest tier of files in the application.

The SiteMapPath control is so easy to work with that it doesn't even require a data source control to hook it up to the `Web.sitemap` file where it infers all its information. All you do is drag-and-drop a SiteMapPath control onto your `MarketsUS.aspx` page. In the end, you should have a page similar to the one shown in Listing 14-2.

Listing 14-2: Using the Web.sitemap file with a SiteMapPath server control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Using the SiteMapPath Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:SiteMapPath ID="Sitemappath1" runat="server">
        </asp:SiteMapPath>
    </form>
</body>
</html>
```

Not much to it, is there? It really is that easy. Run this page and you see the results shown in Figure 14-2.

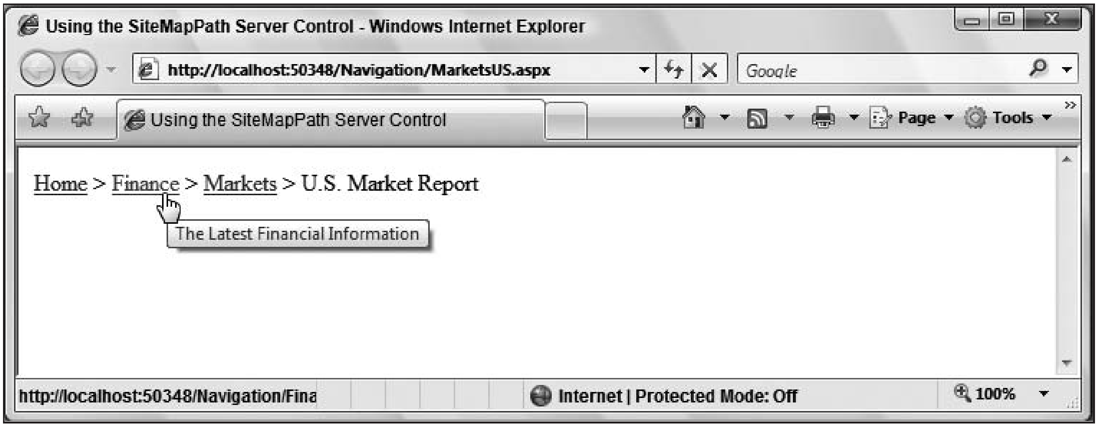

Using the SiteMapPath Server Control - Windows Internet Explorer
http://localhost:50348/Navigation/MarketsUS.aspx
Google
Using the SiteMapPath Server Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

Home > Finance > Markets > U.S. Market Report

Figure 14-2

This screenshot shows that you are on the U.S. Market Report page at `MarketsUS.aspx`. As an end user, you can see that this page is part of the Markets section of the site; Markets, in turn, is part of the Finance section of the site. With breadcrumb navigation, end users who understand the structure of the site and their place in it can quickly select the links to navigate to any location in the site.

If you hover your mouse over the Finance link, you see a tooltip appear after a couple of seconds, as shown in Figure 14-3.


Home > Finance > Markets > U.S. Market Report

The Latest Financial Information

Using the SiteMapPath Server Control - Windows Internet Explorer
http://localhost:50348/Navigation/MarketsUS.aspx
Google
Using the SiteMapPath Server Control
Page
Tools
http://localhost:50348/Navigation/Fina
Internet | Protected Mode: Off
100%

Figure 14-3

This tooltip, which reads `The Latest Financial Information`, comes from the description attribute of the `<siteMapNode>` element in the `Web.sitemap` file.

```
<siteMapNode title="Finance" description="The Latest Financial Information"
 url="Finance.aspx">
```

The SiteMapPath control works automatically requiring very little work on your part. You just add the basic control to your page, and the control automatically creates the breadcrumb navigation you have just seen. However, you can use the properties discussed in the following sections to modify the appearance and behavior of the control.

The PathSeparator Property

One important style property for the SiteMapPath control is the `PathSeparator` property. By default, the SiteMapPath control uses a greater than sign (>) to separate the link elements. You can change this by reassigning a new value to the `PathSeparator` property. Listing 14-3 illustrates the use of this property.

Listing 14-3: Changing the PathSeparator value

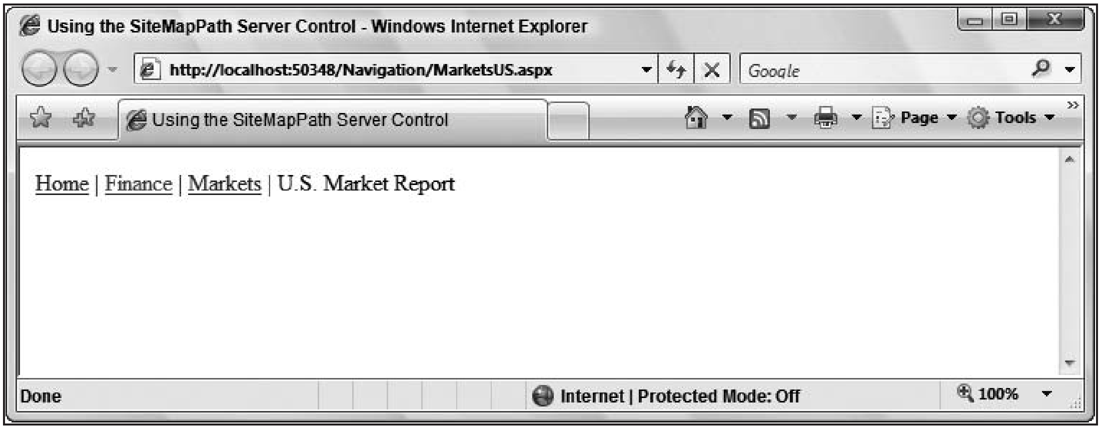
```
<asp:SiteMapPath ID="Sitemappath1" runat="server" PathSeparator=" | ">
</asp:SiteMapPath>
```

Or

```
<asp:SiteMapPath ID="Sitemappath1" runat="server">
   <PathSeparatorTemplate> | </PathSeparatorTemplate>
</asp:SiteMapPath>
```

The SiteMapPath control in this example uses the pipe character (|), which is found above the Enter key. When it is rendered, you get the results shown in Figure 14-4.

As you can see, you can use either the `PathSeparator` attribute or the `<PathSeparatorTemplate>` element within the SiteMapPath control.


Using the SiteMapPath Server Control - Windows Internet Explorer
http://localhost:50348/Navigation/MarketsUS.aspx
Google
Using the SiteMapPath Server Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

Home | Finance | Markets | U.S. Market Report

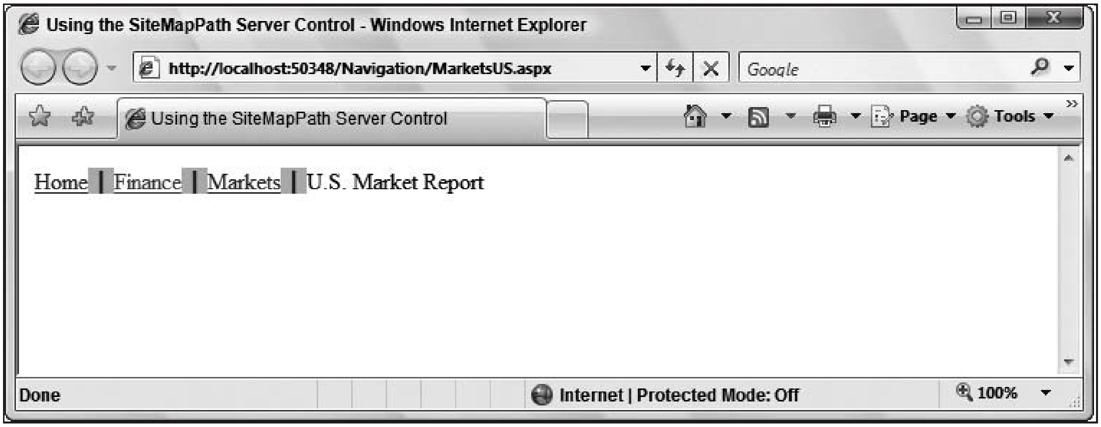
Figure 14-4

With the use of the `PathSeparator` attribute or the `<PathSeparatorTemplate>` element, it is quite easy to specify what you want to use to separate the links in the breadcrumb navigation, but you might also want to give this pipe some visual style as well. You can add a `<PathSeparatorStyle>` node to your SiteMapPath control. An example of this is shown in Listing 14-4.

Listing 14-4: Adding style to the PathSeparator property

```
<asp:SiteMapPath ID="Sitemappath1" runat="server" PathSeparator=" | ">
   <PathSeparatorStyle Font-Bold="true" Font-Names="Verdana" ForeColor="#663333"
    BackColor="#cccc66"></PathSeparatorStyle>
</asp:SiteMapPath>
```

Okay, it may not be pretty, but by using the `<PathSeparatorStyle>` element with the `SiteMapPath` control, we are able to change the visual appearance of the separator elements. The results are shown in Figure 14-5.


Using the SiteMapPath Server Control - Windows Internet Explorer
http://localhost:50348/Navigation/MarketsUS.aspx
Google
Using the SiteMapPath Server Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

Home | Finance | Markets | U.S. Market Report

Figure 14-5

Using these constructs, you can also add an image as the separator, as illustrated in Listing 14-5.

Listing 14-5: Using an image as the separator

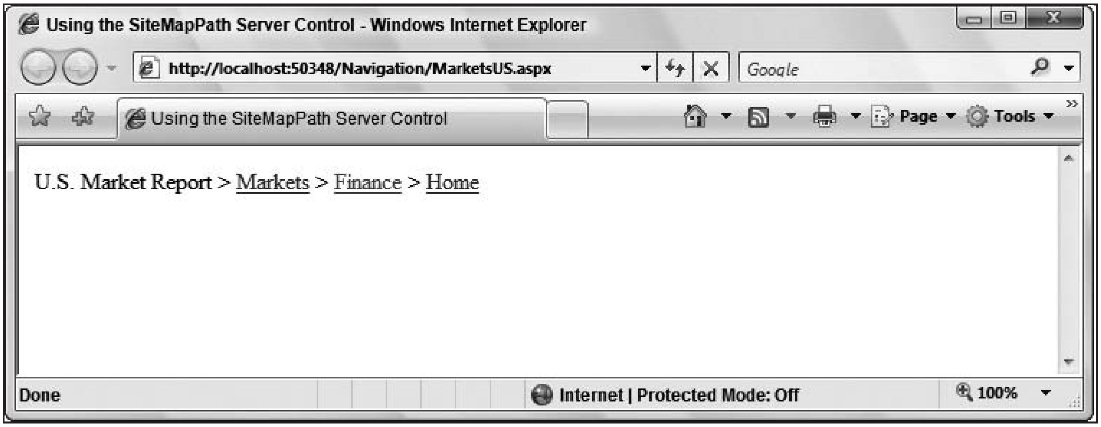
```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Using the SiteMapPath Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:SiteMapPath ID="SiteMapPath1" runat="server">
            <PathSeparatorTemplate>
                <asp:Image ID="Image1" runat="server" ImageUrl="divider.gif" />
            </PathSeparatorTemplate>
        </asp:SiteMapPath>
   </form>
</body>
</html>
```

To utilize an image as the separator between the links, you use the `<PathSeparatorTemplate>` element and place an Image control within it. In fact, you can place any type of control between the navigation links that the SiteMapPath control produces.

The PathDirection Property

Another interesting property to use with the SiteMapPath control is `PathDirection`. This property changes the direction of the links generated in the output. Only two settings are possible for this property: `RootToCurrent` and `CurrentToRoot`.

The Root link is the first link in the display. This is usually the Home page. The Current link is the link for the page currently being displayed. By default, this property is set to `RootToCurrent`. Changing the example to `CurrentToRoot` produces the results shown in Figure 14-6.


Using the SiteMapPath Server Control - Windows Internet Explorer
http://localhost:50348/Navigation/MarketsUS.aspx
Google
Using the SiteMapPath Server Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

U.S. Market Report > Markets > Finance > Home

Figure 14-6

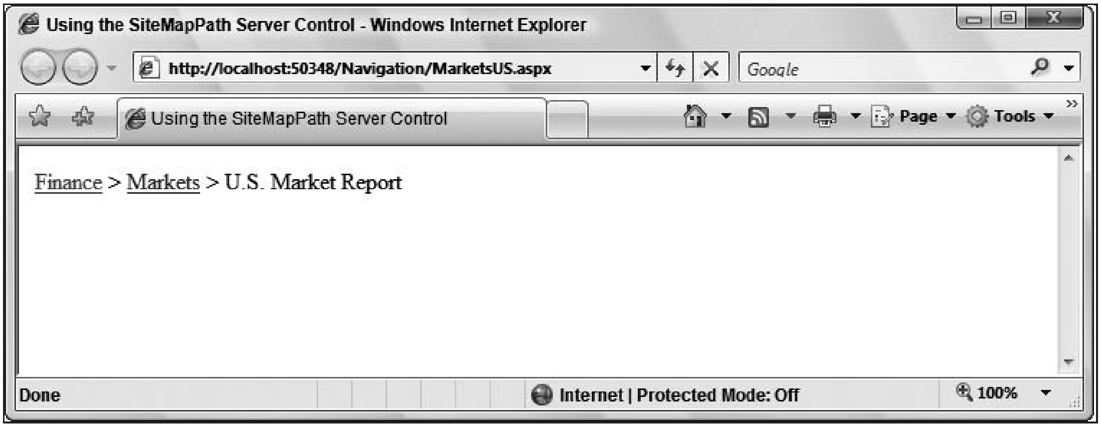
The ParentLevelsDisplayed Property

In some cases, your navigation may go quite deep. You can see on the site map, shown in Listing 14-1, that you go three pages deep, which isn't a big deal. Some of you, however, might be dealing with sites that go quite a number of pages deeper. In these cases, it might be bit silly to use the SiteMapPath control. Doing so would display a huge list of pages.

In a case like this, you can turn to the `ParentLevelsDisplayed` property that is part of the SiteMapPath control. When set, this property displays pages only as deep as specified. Therefore, if you are using the SiteMapPath control with the `Web.sitemap`, as shown in Listing 14-1, and you give the `ParentLevelsDisplayed` property a value of `3`, you don't notice any change to your page. It already displays the path three pages deep. If you change this value to `2`, however, the SiteMapPath control is constructed as follows:

```
<asp:SiteMapPath ID="Sitemappath1" runat="server" ParentLevelsDisplayed="2">
</asp:SiteMapPath>
```

Notice the result of this change in Figure 14-7. The SiteMapPath control shows links only two pages deep and doesn't show the Home page link.


Using the SiteMapPath Server Control - Windows Internet Explorer
http://localhost:50348/Navigation/MarketsUS.aspx
Google
Using the SiteMapPath Server Control
Page
Tools

Finance > Markets > U.S. Market Report

Done
Internet | Protected Mode: Off
100%

Figure 14-7

By default, no limit is set on the number of links shown, so the SiteMapPath control just generates the specified number of links based on what is labeled in the site map file.

The ShowToolTips Property

By default, the SiteMapPath control generates tooltips for each link if a description property is used within the `Web.sitemap` file. Remember, a tooltip is the text that appears onscreen when an end user hovers the mouse over one of the links in the SiteMapPath control. This capability was shown earlier in this chapter.

There may be times when you do not want your SiteMapPath control to show any tooltips for the links that it generates. For these situations, you can actually turn off this capability in a couple of ways. The first way is to omit any description attributes in the `.sitemap` file. If you remove these attributes from the file, the SiteMapPath has nothing to display for the tooltips on the page.

The other way to turn off the display of tooltips is to set the `ShowToolTips` property to `False`, as shown here:

```
<asp:SiteMapPath ID="Sitemappath1" runat="server" ShowToolTips="false">
</asp:SiteMapPath>
```

This turns off the tooltips capability but still enables you to use the description property in the `.sitemap` file. You may still want to use the description attribute because it allows you to keep track of what the links in your file are used for. This is quite advantageous when you are dealing with hundreds or even thousands of links in your application.

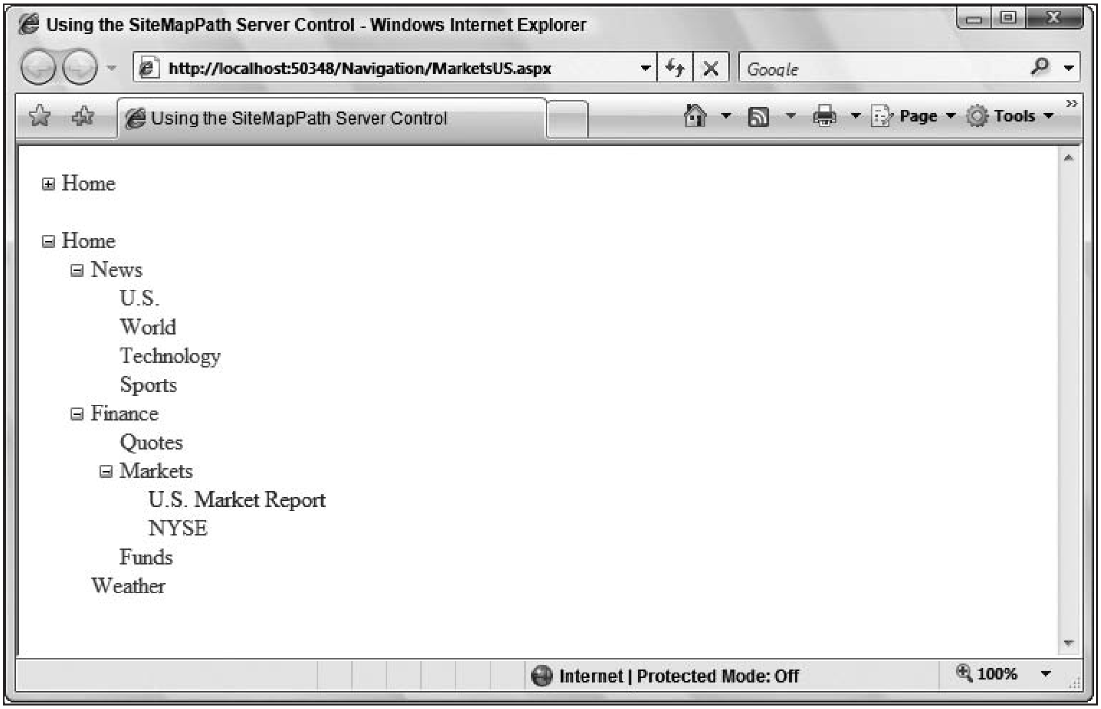
The SiteMapPath Control's Child Elements

You already saw the use of the `<PathSeparatorStyle>` and the `<PathSeparatorTemplate>` child elements for the SiteMapPath control, but additional child elements exist. The following table covers each of the available child elements.

Child Element	Description
`CurrentNodeStyle`	Applies styles to the link in the SiteMapPath navigation for the currently displayed page.
`CurrentNodeTemplate`	Applies a template construction to the link in the SiteMapPath navigation for the currently displayed page.
`NodeStyle`	Applies styles to all links in the SiteMapPath navigation. The settings applied in the `CurrentNodeStyle` or `RootNodeStyle` elements supersede any settings placed here.
`NodeTemplate`	Applies a template construction to all links in the SiteMapPath navigation. The settings applied in the `CurrentNodeTemplate` or `RootNodeTemplate` elements supersede any settings placed here.
`PathSeparatorStyle`	Applies styles to the link dividers in the SiteMapPath navigation.
`PathSeparatorTemplate`	Applies a template construction to the link dividers in the SiteMapPath navigation.
`RootNodeStyle`	Applies styles to the first link (the root link) in the SiteMapPath navigation.
`RootNodeTemplate`	Applies a template construction to the first link in the SiteMapPath navigation.

TreeView Server Control

The TreeView server control is a rich server control for rendering a hierarchy of data, so it is quite ideal for displaying what is contained in your `.sitemap` file. Figure 14-8 shows you how it displays the contents of the site map (again from Listing 14-1) that you have been working with thus far in this chapter. This figure first shows a completely collapsed TreeView control at the top of the screen; the second TreeView control has been completely expanded.


Using the SiteMapPath Server Control - Windows Internet Explorer
http://localhost:50348/Navigation/MarketsUS.aspx
Google
Using the SiteMapPath Server Control
Page
Tools
Internet | Protected Mode: Off
100%

⊞ Home

⊟ Home
  ⊟ News
    U.S.
    World
    Technology
    Sports
  ⊟ Finance
    Quotes
    ⊟ Markets
      U.S. Market Report
      NYSE
    Funds
  Weather

Figure 14-8

This control can dynamically load the nodes to be displayed as they are selected by the expandable and collapsible framework of the control. If the control can render the TreeView output along with some client-side script, the control can make a call back to the server if someone expands one of the nodes in the control to get the subnodes of the selected item. This is ideal if your site navigation system is large. In this case, loading nodes of the TreeView control dynamically greatly helps performance. One of the great features of this postback capability is it is done under the covers and does not require the ASP.NET page to be completely refreshed. Of course, this capability is there only if the browser accepts the client-side code that goes along with the TreeView control. If the browser does not, the control knows this and renders only what is appropriate (pulling all the information that is required of the entire TreeView control). It only performs these JavaScript-based postbacks for those clients who can work with this client-side script.

You can definitely see this in action if you run the TreeView control on a page that is being monitored by an HTTP sniffer of some kind to monitor the traffic moving across the wire.

> *I recommend Fiddler by Eric Lawrence of Microsoft that is freely downloadable on the Internet at* `fiddlertool.com`.

If your browser allows client-side script and you expand one of the expandable nodes of the TreeView control, your HTTP request will be similar to the following:

```
POST /Navigation/Default.aspx HTTP/1.1
Accept: */*
Accept-Language: en-us
Referrer: http://localhost:1882/Navigation/Default.aspx
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1;
.NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.04506;
.NET CLR 3.5.20404; .NET CLR 1.1.4322)
Host: localhost:1882
Content-Length: 904
Proxy-Connection: Keep-Alive
Pragma: no-cache

__EVENTTARGET=&__EVENTARGUMENT=&TreeView1_ExpandState=c&TreeView1_SelectedNode=Tree
View1t0&TreeView1_PopulateLog=&__VIEWSTATE=%2FwEPDwUKLTY0ODk0OTE2Mg9kFgICBA9kFgICAw
88KwAJAgAPFggeDU5ldmVyRXhwYW5kZWRkHgtfIURhdGFCb3VuZGceDFNlbGVjdGVkTm9kZQULVHJlZVZpZ
XcxdDAeCUxhc3RJbmRleAIBZAgUKwACBQMwOjAUKwACFhAeBFRleHQFBEhvbWUeBVZhbHVlBQRIb21lHgtO
YXZpZ2F0ZVVybAUYL05hdmlnYXRpb24vRGVmYXVsdC5hc3B4HgdUb29sVGlwBQlIb21lIFBhZ2UeCERhdGF
QYXRoBRgvbmF2aWdhdGlvbi9kZWZhdWx0LmFzcHgeCURhdGFCb3VuZGceCFNlbGVjdGVkZx4QUG9wdWxhdG
VPbkRlbWFuZGdkZBgBBR5fX0NvbnRyb2xzUmVxdWlyZVBvc3RCYWNrS2V5X18WAwURTG9naW4xJFJlbWVtY
mVyTWUFF0xvZ2luMSRMb2dpbkltYWdlQnV0dG9uBQlUcmVlVmlldzFtwszVpUMxFTDtpERnNjgEIkWWbg%3
D%3D&Login1$UserName=&Login1$Password=&__CALLBACKID=TreeView1&__CALLBACKPARAM=0%7C1
%7Ctft%7C4%7CHome24%7C%2Fnavigation%2Fdefault.aspxHome&__EVENTVALIDATION=%2FwEWBgKg
8Yn8DwKUvNa1DwL666vYDAKC0q%2BkBgKnz4ybCAKn5fLxBaSy6WQwPagNZsHisWRoJfuiopOe
```

The response from your ASP.NET application will *not* be the entire page that holds the TreeView control, but instead it is a small portion of HTML that is used by a JavaScript method on the page and is loaded into the TreeView control dynamically. A sample response is illustrated here:

```
HTTP/1.1 200 OK
Server: ASP.NET Development Server/8.0.0.0
Date: Sat, 11 Feb 2008 17:55:02 GMT
X-AspNet-Version: 2.0.50727
Cache-Control: private, no-store
Content-Type: text/html; charset=utf-8
Content-Length: 1756
Connection: Close

112|/wEWCgKg8Yn8DwKUvNa1DwL666vYDAKC0q+kBgKnz4ybCAKn5fLxBQKAgtPaBALEmcbhCgK8nZDfCAL
M/ZK8AR/nFcl4nlPgp6HcFlU6YiFBfoNM14|nn|<div id="TreeView1n6Nodes"
style="display:none;">
 <table cellpadding="0" cellspacing="0" style="border-width:0;">
        <tr>
                <td><div style="width:20px;height:1px"></div></td><td><div
                 style="width:20px;height:1px"><img
                 src="/Navigation/WebResource.axd?d=GOWKLfnbFU9fYyy
                 PCMT8DIfngU4PXeMiAHxJNuXB-tU1&amp;t=632662834831594592" alt="" />
                </div></td><td><div style="width:20px;height:1px"><img
                 src="/Navigation/WebResource.axd?d=GOWKLfnbFU9fYyyPCMT8DIfngU
                 4PXeMiAHxJNuXB-tU1&amp;t=632662834831594592" alt="" />
                </div></td><td><img
```

```
                  src="/Navigation/WebResource.axd?d=GOWKLfnbFU9fYyy
                  PCMT8DCXmyNCWX5x-n_pSXFIW2qE1&amp;t=632662834831594592"
                  alt="" /></td><td style="white-space:nowrap;">
                 <a href="/Navigation/MarketsUSasdf.aspx"
                  title="Looking at the U.S. Market" id="TreeView1t12"
                  style="text-decoration:none;">U.S. Market Report</a></td>
         </tr>
  </table><table cellpadding="0" cellspacing="0" style="border-width:0;">
         <tr>
                  <td><div style="width:20px;height:1px"></div></td><td><div
                  style="width:20px;height:1px"><img
                  src="/Navigation/WebResource.axd?d=GOWKLfnbFU9fYyyPCMT8DI
                  fngU4PXeMiAHxJNuXB-tU1&amp;t=632662834831594592" alt="" />
                 </div></td><td><div style="width:20px;height:1px"><img
                  src="/Navigation/WebResource.axd?d=GOWKLfnbFU9fYyyPCMT8DI
                  fngU4PXeMiAHxJNuXB-tU1&amp;t=632662834831594592" alt="" />
                 </div></td><td><img
                  src="/Navigation/WebResource.axd?d=GOWKLfnbFU9fY
                  yyPCMT8DGyYai5iS-79vjeYzdeJoiI1&amp;t=632662834831594592"
                  alt="" />
                 </td><td style="white-space:nowrap;">
                 <a href="/Navigation/NYSE.aspx" title="The New York Stock Exchange"
                  id="TreeView1t13" style="text-decoration:none;">NYSE</a></td>
        </tr>
  </table>
 </div>
```

This postback capability is rather powerful, but if you want to disable it (even for browsers that can handle it), you just set the `PopulateNodesFromClient` attribute to `false` in the TreeView control (the default value is `true`).

The TreeView control is quite customizable; but first, take a look at how to create a default version of the control using the `.sitemap` file from Listing 14-1. For this example, continue to use the `MarketsUS.aspx` page you created earlier.

The first step is to create a SiteMapDataSource control on the page. When working with the TreeView control that displays the contents of your `.sitemap` file, you must apply one of these data source controls. The TreeView control doesn't just bind to your site map file automatically as the SiteMapPath control does.

After a basic SiteMapDataSource control is in place, position a TreeView control on the page and set the `DataSourceId` property to `SiteMapDataSource1`. When you have finished, your code should look like Listing 14-6.

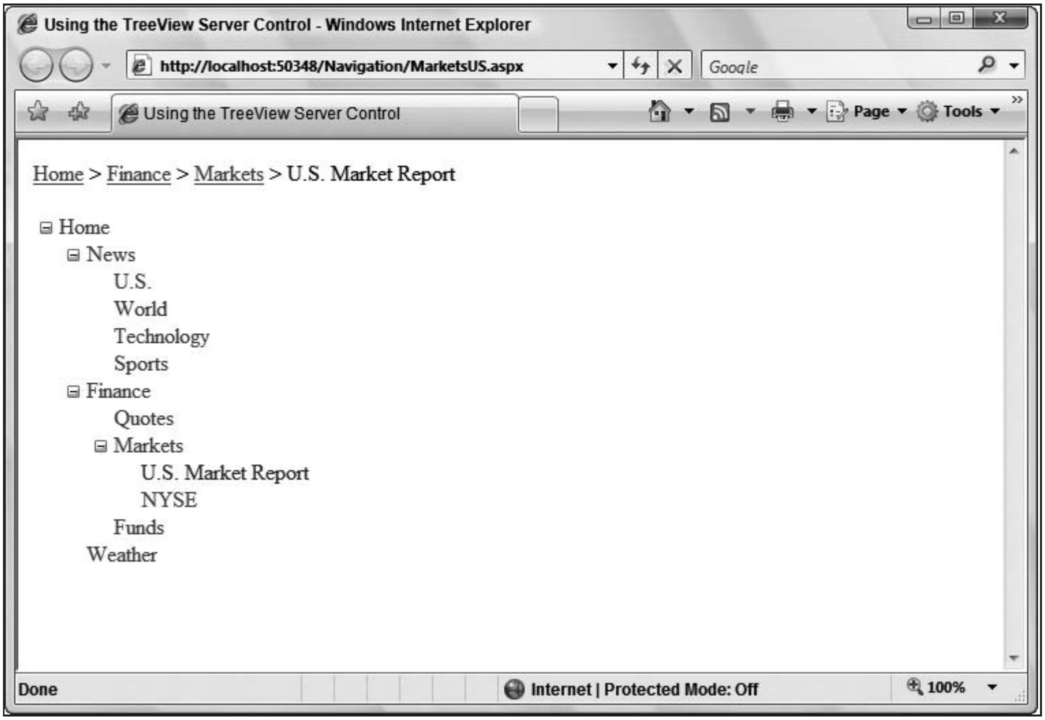
Listing 14-6: A basic TreeView control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Using the TreeView Server Control</title>
</head>
```

Continued

```
<body>
    <form id="form1" runat="server">
        <asp:SiteMapPath ID="SiteMapPath1" runat="server">
        </asp:SiteMapPath>
        <br /><p>
        <asp:TreeView ID="TreeView1" runat="server"
         DataSourceID="SiteMapDataSource1">
        </asp:TreeView>
        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" /></p>
    </form>
</body>
</html>
```

After the page is run and the TreeView control is expanded, the results are displayed as shown in Figure 14-9.


Using the TreeView Server Control - Windows Internet Explorer
http://localhost:50348/Navigation/MarketsUS.aspx
Google
Using the TreeView Server Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

Home > Finance > Markets > U.S. Market Report

- ⊟ Home
  - ⊟ News
    - U.S.
    - World
    - Technology
    - Sports
  - ⊟ Finance
    - Quotes
    - ⊟ Markets
      - U.S. Market Report
      - NYSE
    - Funds
  - Weather

Figure 14-9

This is a very basic TreeView control. The great thing about this control is that it allows for a high degree of customization and even gives you the capability to use some predefined styles that come prepackaged with ASP.NET 3.5.

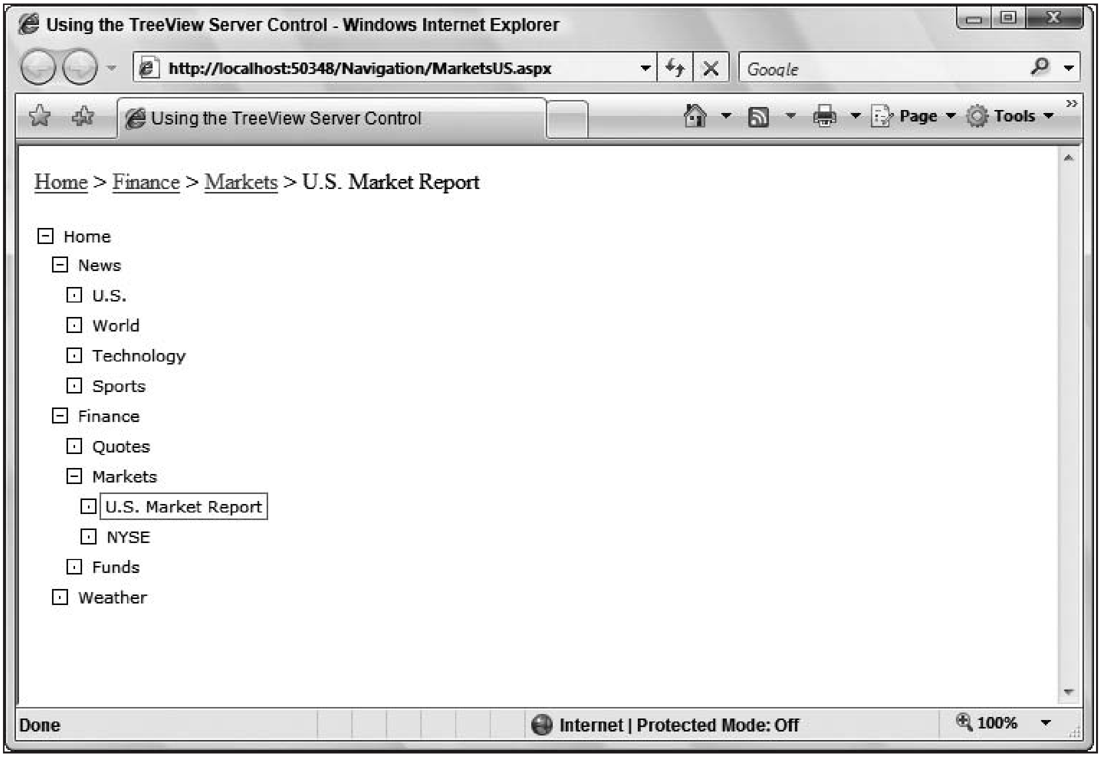
Identifying the TreeView Control's Built-In Styles

As stated, the TreeView control does come with a number of pre-built styles right out of the box. The best way to utilize these predefined styles is to do so from the Design view of your page. By clicking on the arrow located in the upper right section of the server control in the Design view in Visual Studio 2008, you will find the Auto Format option. Click this option and a number of styles become available to you. Selecting one of these styles changes the code of your TreeView control to adapt to that chosen style. For instance, if you choose `MSDN` from the list of options, the simple one-line TreeView control you created is converted to what is shown in Listing 14-7.

Listing 14-7: A TreeView control with the MSDN style applied to it

```
<asp:TreeView ID="TreeView1" runat="server" DataSourceID="SiteMapDataSource1"
 ImageSet="Msdn" NodeIndent="10">
   <ParentNodeStyle Font-Bold="False" />
   <HoverNodeStyle BackColor="#CCCCCC" BorderColor="#888888" BorderStyle="Solid"
    Font-Underline="True" />
   <SelectedNodeStyle BackColor="White" BorderColor="#888888" BorderStyle="Solid"
    BorderWidth="1px" Font-Underline="False" HorizontalPadding="3px"
    VerticalPadding="1px" />
   <NodeStyle Font-Names="Verdana" Font-Size="8pt" ForeColor="Black"
    HorizontalPadding="5px" NodeSpacing="1px" VerticalPadding="2px" />
</asp:TreeView>
```

As you can see, if you use these built-in styles, it is not too difficult to completely change the look and feel of the TreeView control. When this bit of code is run, you get the results shown in Figure 14-10.


Using the TreeView Server Control - Windows Internet Explorer
http://localhost:50348/Navigation/MarketsUS.aspx
Google
Using the TreeView Server Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

Home > Finance > Markets > U.S. Market Report

- ⊟ Home
  - ⊟ News
    - ⊡ U.S.
    - ⊡ World
    - ⊡ Technology
    - ⊡ Sports
  - ⊟ Finance
    - ⊡ Quotes
    - ⊟ Markets
      - ⊡ U.S. Market Report
      - ⊡ NYSE
    - ⊡ Funds
  - ⊡ Weather

Figure 14-10

Examining the Parts of the TreeView Control

To master working with the TreeView control, you must understand the terminology used for each part of the hierarchical tree that is created by the control.

First, every element or entry in the TreeView control is called a *node*. The uppermost node in the hierarchy of nodes is the *root node*. It is possible for a TreeView control to have multiple root nodes. Any node, including the root node, is also considered a *parent node* if it has any nodes that are directly under it in the hierarchy of nodes. The nodes directly under this parent node are referred to as *child nodes*. Each parent node can have one or more child nodes. Finally, if a node contains no child nodes, it is referred to as a *leaf node*.

The following is based on the site map shown earlier and details the use of this terminology:

```
Home - Root node, parent node
   News - Parent node, child node
      U.S. - Child node, leaf node
      World - Child node, leaf node
      Technology - Child node, leaf node
      Sports - Child node, leaf node
   Finance - Parent node, child node
      Quotes - Child node, leaf node
      Markets - Parent node, child node
         U.S. Market Report - Child node, leaf node
         NYSE - Child node, leaf node
      Funds - Child node, leaf node
   Weather - Child node, leaf node
```

From this listing, you can see what each node is and how it is referred in the hierarchy of nodes. For instance, the `U.S. Market Report` node is a leaf node — meaning that it doesn't have any child nodes associated with it. However, it is also a child node to the `Markets` node, which is a parent node to the `U.S. Market Report` node. If you are working with the `Markets` node directly, it is also a child node to the `Finance` node, which is its parent node. The main point to take away from all this is that each node in the site map hierarchy has a relationship to the other nodes in the hierarchy. You must understand these relationships because you can programmatically work with these nodes (as will be demonstrated later in this chapter), and the methods used for working with them include terms such as `RootNode`, `CurrentNode` and `ParentNode`.

Binding the TreeView Control to an XML File

You are not limited to working with just a `.sitemap` file in order to populate the nodes of your TreeView controls. You have many ways to get this done. One cool way is to use the XmlDataSource control (instead of the SiteMapDataSource control) to populate your TreeView controls from your XML files.

For an example of this, create a hierarchical list of items in an XML file called `Hardware.xml`. An example of this is shown in Listing 14-8.

Listing 14-8: Hardware.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Hardware>
    <Item Category="Motherboards">
        <Option Choice="Asus" />
        <Option Choice="Abit" />
    </Item>
    <Item Category="Memory">
        <Option Choice="128mb" />
        <Option Choice="256mb" />
        <Option Choice="512mb" />
    </Item>
    <Item Category="HardDrives">
        <Option Choice="40GB" />
        <Option Choice="80GB" />
        <Option Choice="100GB" />
    </Item>
    <Item Category="Drives">
        <Option Choice="CD" />
        <Option Choice="DVD" />
        <Option Choice="DVD Burner" />
    </Item>
</Hardware>
```

As you can see, this list is not meant to be used for site navigation purposes, but instead for allowing the end user to make a selection from a hierarchical list of options. This XML file is divided into four categories of available options: `Motherboards`, `Memory`, `HardDrives`, and `Drives`. To bind your TreeView control to this XML file, use an XmlDataSource control that specifies the location of the XML file you are going to use. Then within the TreeView control itself, use the `<asp:TreeNodeBinding>` element to specify which elements to bind in the XML file to populate the nodes of the TreeView control. This is illustrated in Listing 14-9.

Listing 14-9: Binding a TreeView control to the Hardware.xml file

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Latest Hardware</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:TreeView ID="TreeView1" runat="server" DataSourceID="XmlDataSource1">
            <DataBindings>
                <asp:TreeNodeBinding DataMember="Hardware"
                 Text="Computer Hardware" />
                <asp:TreeNodeBinding DataMember="Item" TextField="Category" />
```

Continued

```
                <asp:TreeNodeBinding DataMember="Option" TextField="Choice" />
            </DataBindings>
        </asp:TreeView>
        <asp:XmlDataSource ID="XmlDataSource1" runat="server"
         DataFile="Hardware.xml">
        </asp:XmlDataSource>
    </form>
</body>
</html>
```

The first item to look at is the `<asp:XmlDataSource>` control. It is just as simple as the previous `<asp:SiteMapDataSource>` control, but it points at the `Hardware.xml` file using the `DataFile` property.

The next step is to create a TreeView control that binds to this particular XML file. You can bind a default TreeView control directly to the XmlDataSource control such as this:

```
<asp:TreeView ID="TreeView1" runat="server" DataSourceId="XmlDataSource1" />
```

Doing this, you get the *incorrect* result shown in Figure 14-11.

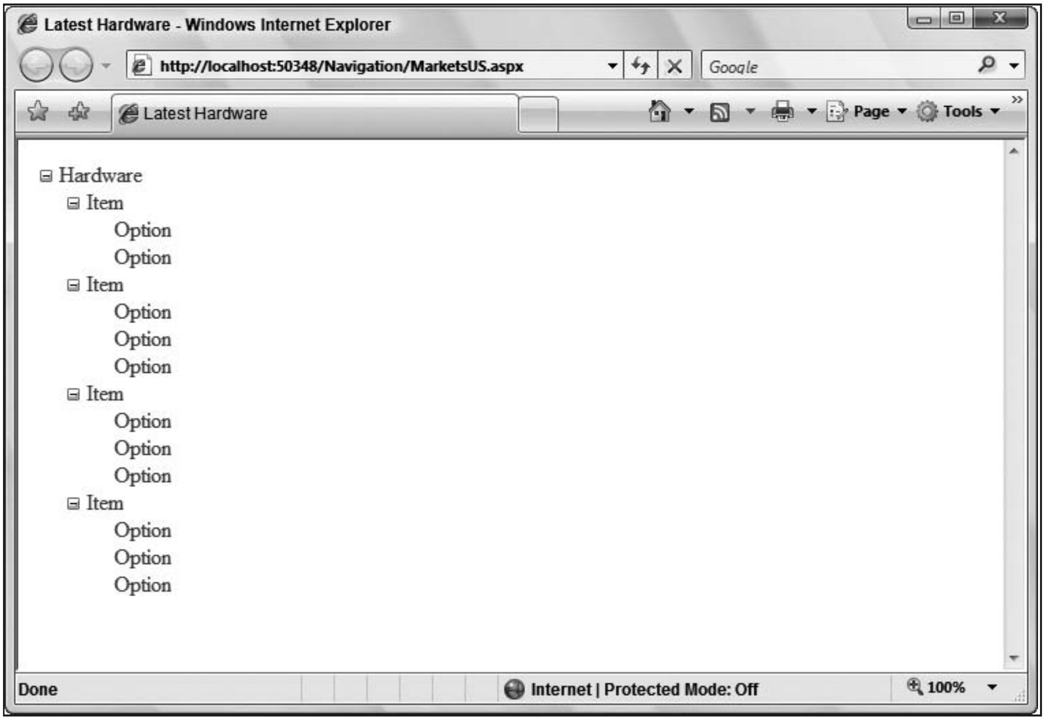
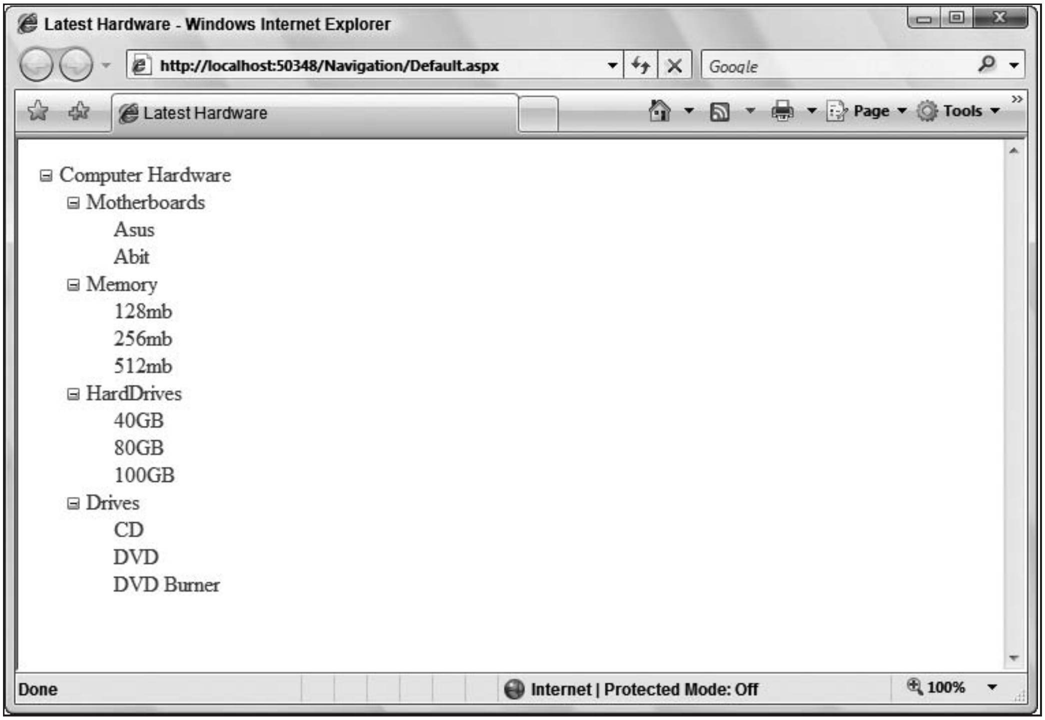

Latest Hardware - Windows Internet Explorer
http://localhost:50348/Navigation/MarketsUS.aspx
Google
Latest Hardware
Page
Tools
Hardware
Item
Option
Option
Item
Option
Option
Option
Item
Option
Option
Option
Item
Option
Option
Option
Done
Internet | Protected Mode: Off
100%

Figure 14-11

As you can see, the TreeView control binds just fine to the `Hardware.xml` file, but looking at the nodes within the TreeView control, you can see that it is simply displaying the names of the actual XML elements from the file itself. Because this isn't what you want, you specify how to bind to the XML file with the use of the `<DataBindings>` element within the TreeView control.

The `<DataBindings>` element encapsulates one or more `TreeNodeBinding` objects. Two of the more important available properties of a `TreeNodeBinding` object are the `DataMember` and `TextField` properties. The `DataMember` property points to the name of the XML element that the TreeView control should look for. The `TextField` property specifies the XML attribute that the TreeView should look for in that particular XML element. If you do this correctly, using the `<DataBindings>` construct, you get the result shown in Figure 14-12.


Latest Hardware - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
Latest Hardware
Page
Tools
Done
Internet | Protected Mode: Off
100%

- ⊟ Computer Hardware
  - ⊟ Motherboards
    - Asus
    - Abit
  - ⊟ Memory
    - 128mb
    - 256mb
    - 512mb
  - ⊟ HardDrives
    - 40GB
    - 80GB
    - 100GB
  - ⊟ Drives
    - CD
    - DVD
    - DVD Burner

Figure 14-12

You can also see from Listing 14-9 that you can override the text value of the root node from the XML file, `<Hardware>`, and have it appear as `Computer Hardware` in the TreeView control, as follows:

```
<asp:TreeNodeBinding DataMember="Hardware" Text="Computer Hardware" />
```

Selecting Multiple Options in a TreeView

As stated earlier, the TreeView control is not meant to be used exclusively for navigation purposes. You can use it for all sorts of things. In many cases, you can present a hierarchical list from which you want the end user to select one or more items.

One great built-in feature of the TreeView control is the capability to put check boxes next to nodes within the hierarchical items in the list. These boxes enable end users to make multiple selections. The TreeView control contains a property called `ShowCheckBoxes` that can be used to create check boxes next to many different types of nodes within a list of items.

The available values for the `ShowCheckBoxes` property are discussed in the following table.

Value	Description
`All`	Applies check boxes to each and every node within the TreeView control.
`Leaf`	Applies check boxes to only the nodes that have no additional child elements.
`None`	Applies no check boxes to any node within the TreeView control.
`Parent`	Applies check boxes to only the nodes considered parent nodes within the TreeView control. A parent node has at least one child node associated with it.
`Root`	Applies a check box to any root node contained within the TreeView control.

When working with the `ShowCheckBoxes` property, you can set it declaratively in the control itself, as follows:

```
<asp:TreeView ID="Treeview1" runat="server" Font-Underline="false"
 DataSourceID="XmlDataSource1" ShowCheckBoxes="Leaf">
    ...
</asp:TreeViewTreeView>
```

Or you can set it programmatically by using the following code:

VB

```
TreeView1.ShowCheckBoxes = TreeNodeTypes.Leaf
```

C#

```
TreeView1.ShowCheckBoxes = TreeNodeTypes.Leaf;
```

For an example of using check boxes with the TreeView control, let's continue to expand on the computer hardware example from Listing 14-9. Create a hierarchical list that enables people to select multiple items from the list in order to receive additional information about them. Listing 14-10 shows an example of this.

Listing 14-10: Applying check boxes next to the leaf nodes within the hierarchical list of nodes

VB

```
<%@ Page Language="VB" %>

<script runat="server">
   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        If TreeView1.CheckedNodes.Count > 0 Then
            Label1.Text = "We are sending you information on:<p>"

            For Each node As TreeNode In TreeView1.CheckedNodes
                Label1.Text += node.Text & " " & node.Parent.Text & "<br>"
            Next
        Else
            Label1.Text = "You didn't select anything. Sorry!"
        End If
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Latest Hardware</title>
</head>
<body>
    <form runat="server">
    Please select the items you are interested in:
        <p>
        <asp:TreeView ID="TreeView1" runat="server" Font-Underline="False"
         DataSourceID="XmlDataSource1" ShowCheckBoxes="Leaf">
            <DataBindings>
                <asp:TreeNodeBinding DataMember="Hardware"
                 Text="Computer Hardware" />
                <asp:TreeNodeBinding DataMember="Item" TextField="Category" />
                <asp:TreeNodeBinding DataMember="Option" TextField="Choice" />
            </DataBindings>
        </asp:TreeView>
        <p>
        <asp:Button ID="Button1" runat="server" Text="Submit Choices"
         OnClick="Button1_Click" />
        </p>
        <asp:XmlDataSource ID="XmlDataSource1" runat="server"
         DataFile="Hardware.xml">
        </asp:XmlDataSource>
       </p>
       <asp:Label ID="Label1" runat="Server" />
    </form>
</body>
</html>
```

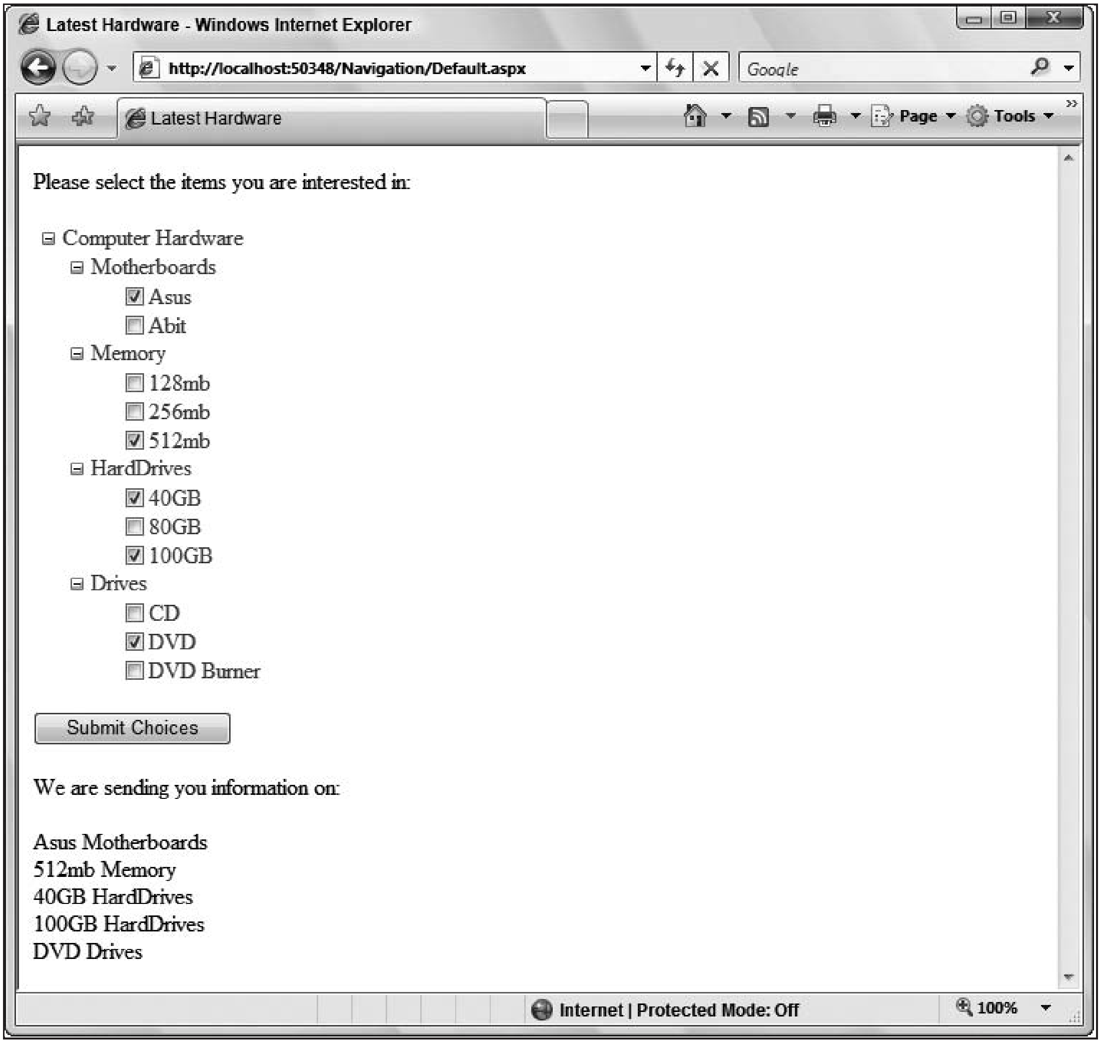
C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, System.EventArgs e)
    {
        if (TreeView1.CheckedNodes.Count > 0)
        {
            Label1.Text = "We are sending you information on:<p>";
            foreach (TreeNode node in TreeView1.CheckedNodes)
            {
                Label1.Text += node.Text + " " + node.Parent.Text + "<br>";
     }
        }
        else
        {
            Label1.Text = "You didn't select anything. Sorry!";
        }
    }
</script>
```

In this example, you first set the `ShowTextBoxes` property to `Leaf`, meaning that you are interested in having check boxes appear only next to items in the TreeView control that do not contain any child nodes. The items with check boxes next to them should be the last items that can be expanded in the hierarchical list.

After this property is set, you then work with the items that are selected by the end user in the `Button1_Click` event. The first thing you should check is whether any selection at all was made:

```
If TreeView1.CheckedNodes.Count > 0 Then
   ...
End If
```


Latest Hardware - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
Latest Hardware
Page
Tools
Internet | Protected Mode: Off
100%

Please select the items you are interested in:

- ⊟ Computer Hardware
  - ⊟ Motherboards
    - ☑ Asus
    - ☐ Abit
  - ⊟ Memory
    - ☐ 128mb
    - ☐ 256mb
    - ☑ 512mb
  - ⊟ HardDrives
    - ☑ 40GB
    - ☐ 80GB
    - ☑ 100GB
  - ⊟ Drives
    - ☐ CD
    - ☑ DVD
    - ☐ DVD Burner

Submit Choices

We are sending you information on:

Asus Motherboards
512mb Memory
40GB HardDrives
100GB HardDrives
DVD Drives

Figure 14-13

In this case, the number of checked nodes on the postback needs to be greater than zero, meaning that at least one was selected. If so, you can execute the code within the `If` statement. The `If` statement then proceeds to populate the Label control that is on the page. To populate the Label control with data from the selected nodes, you use a `For Each` statement, as shown here:

```
For Each node As TreeNode In TreeView1.CheckedNodes
   ...
Next
```

This works with an instance of a `TreeNode` object and checks each `TreeNode` object within the `TreeView1` collection of checked nodes.

For each node that is checked, you grab the nodes `Text` value and the `Text` value of this node's parent node to further populate the Label control, as follows:

```
Label1.Text += node.Text & " " & node.Parent.Text & "<br>"
```

In the end, you get a page that produces the results shown in Figure 14-13.

Specifying Custom Icons in the TreeView Control

The TreeView control allows for a high degree of customization. You saw earlier in the chapter that you were easily able to customize the look-and-feel of the TreeView control by specifying one of the built-in styles. Applying one of these styles dramatically changes the appearance of the control. One of the most noticeable changes concerns the icons used for the nodes within the TreeView control. Although it is not as easy as just selecting one of the styles built into the TreeView control, you can apply your own icons to be used for the nodes within the hierarchical list of nodes.

The TreeView control contains the properties discussed in the following table. These properties enable you to specify your own images to use for the nodes of the control.

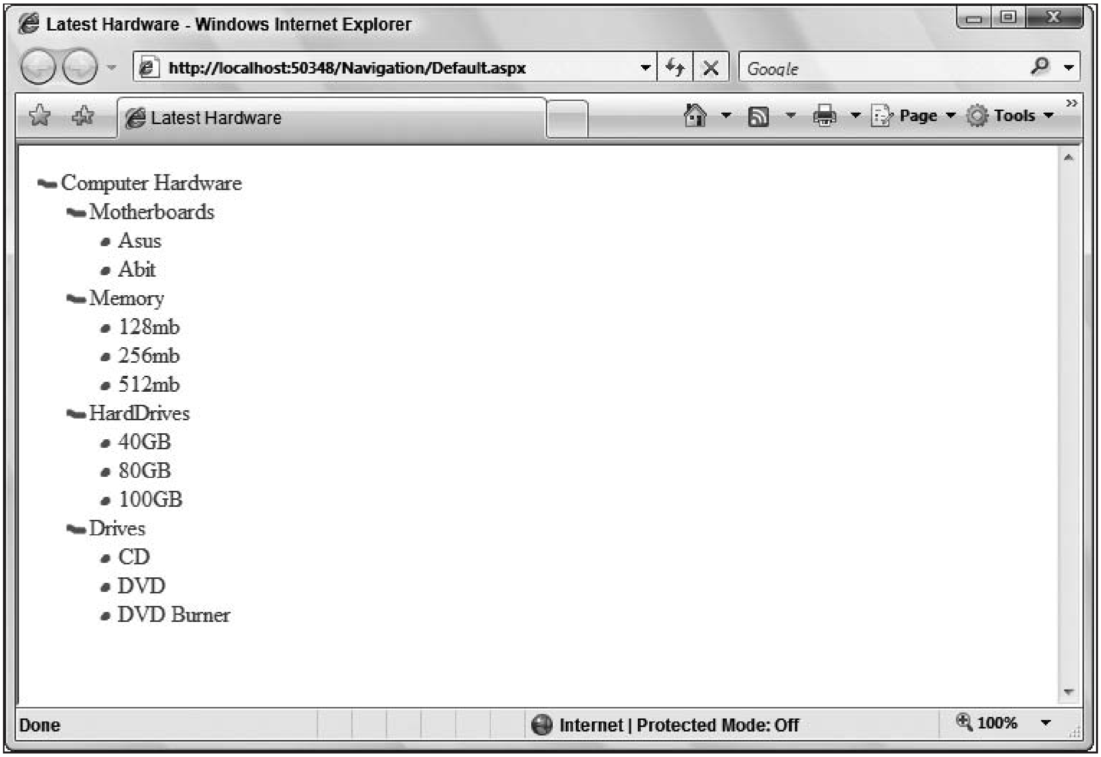
Property	**Description**
`CollapseImageUrl`	Applies a custom image next to nodes that have been expanded to show any of their child nodes and have the capability of being collapsed.
`ExpandImageUrl`	Applies a custom image next to nodes that have the capability of being expanded to display their child nodes.
`LeafNodeStyle-ImageUrl`	Applies a custom image next to a node that has no child nodes and is last in the hierarchical chain of nodes.
`NoExpandImageUrl`	Applies a custom image to nodes that, for programmatic reasons, cannot be expanded or to nodes that are leaf nodes. This is primarily used for spacing purposes to align leaf nodes with their parent nodes.
`ParentNodeStyle-ImageUrl`	Applies a custom image only to the parent nodes within the TreeView control.
`RootNodeStyle-ImageUrl`	Applies a custom image next to only the root nodes within the TreeView control.

Listing 14-11 shows an example of these properties in use.

Listing 14-11: Applying custom images to the TreeView control

```
<asp:TreeView ID="TreeView1" runat="server" Font-Underline="False"
 DataSourceId="XmlDataSource1"
 CollapseImageUrl="Images/CollapseImage.gif"
 ExpandImageUrl="Images/ExpandImage.gif"
 LeafNodeStyle-ImageUrl="Images/LeafImage.gif">
   <DataBindings>
      <asp:TreeNodeBinding DataMember="Hardware" Text="Computer Hardware" />
      <asp:TreeNodeBinding DataMember="Item" TextField="Category" />
      <asp:TreeNodeBinding DataMember="Option" TextField="Choice" />
   </DataBindings>
</asp:TreeView>
```

Specifying these three images to precede the nodes in your control overrides the default values of using a plus (+) sign and a minus (–) sign for the expandable and collapsible nodes. It also overrides simply using an image for any leaf nodes when by default nothing is used. Using the code from Listing 14-11, you get something similar to the results illustrated in Figure 14-14 (depending on the images you use, of course).


Latest Hardware - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
Latest Hardware
Page
Tools
Done
Internet | Protected Mode: Off
100%

- Computer Hardware
  - Motherboards
    - Asus
    - Abit
  - Memory
    - 128mb
    - 256mb
    - 512mb
  - HardDrives
    - 40GB
    - 80GB
    - 100GB
  - Drives
    - CD
    - DVD
    - DVD Burner

Figure 14-14

Specifying Lines Used to Connect Nodes

Because the TreeView control shows a hierarchical list of items to the end user, you sometimes want to show the relationship between these hierarchical items more explicitly than it is shown by default with the TreeView control. One possibility is to show line connections between parent and child nodes within the display. Simply set the `ShowLines` property of the TreeView control to `True` (by default, this property is set to `False`):

```
<asp:TreeView ID="TreeView1" runat="server" Font-Underline="False"
 DataSourceId="XmlDataSource1" ShowCheckBoxes="Leaf" ShowLines="True">
   ...
</asp:TreeViewTreeView>
```

This code gives the result shown in Figure 14-15.

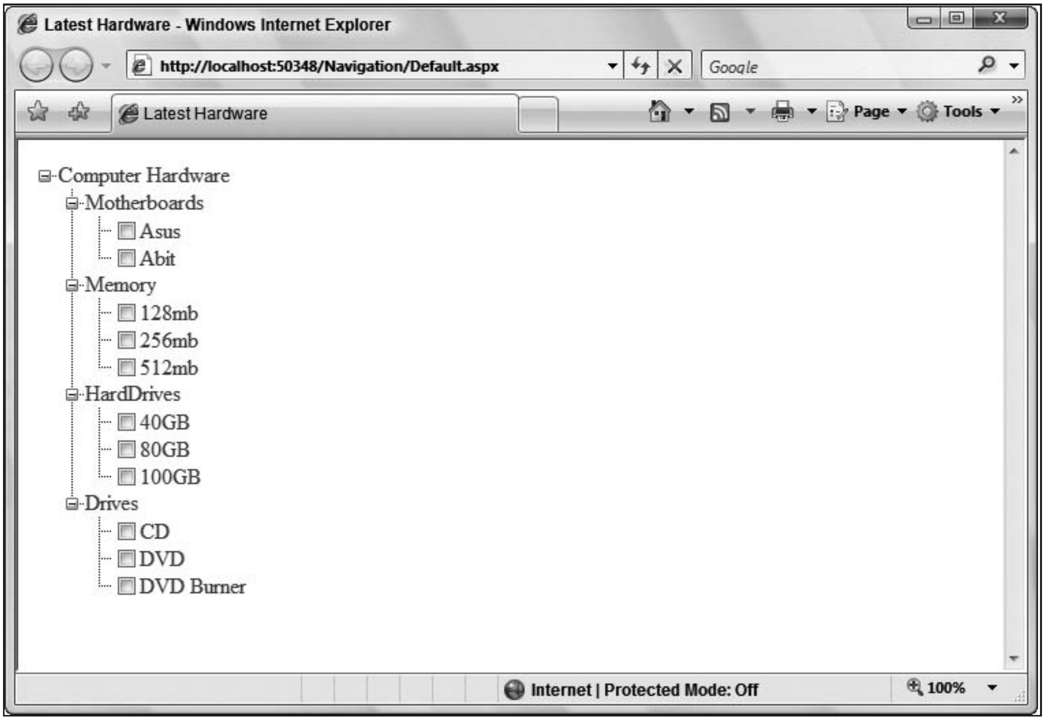
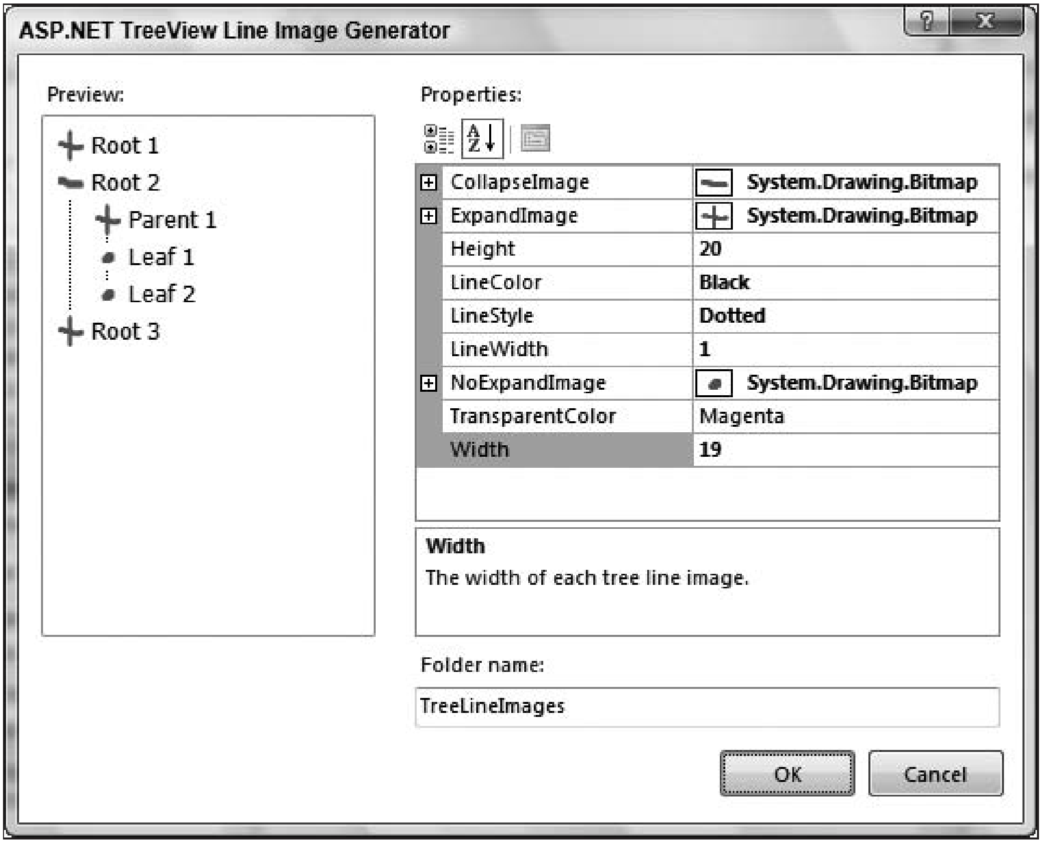

Latest Hardware - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
Latest Hardware
Page
Tools
Computer Hardware
Motherboards
Asus
Abit
Memory
128mb
256mb
512mb
HardDrives
40GB
80GB
100GB
Drives
CD
DVD
DVD Burner
Internet | Protected Mode: Off
100%

Figure 14-15

If the `ShowLines` property is set to `True`, you can also define your own lines and images within the TreeView control. This is quite easy to do because Visual Studio 2008 provides you with an ASP.NET TreeView Line Image Generator tool. This tool enables you to visually design how you want the lines and corresponding expanding and collapsing images to appear. After you have it set up as you want, the tool then creates all the necessary files for any of your TreeView controls to use.

To get at the tool, move to the Design view of your file and click the smart tag for the TreeView control that is on your page. Here you find the option Customize Line Images. You will not see this option unless the Show Lines check box is checked. Click this and you are presented with the ASP.NET TreeView Line Generator dialog (shown in Figure 14-16).



ASP.NET TreeView Line Image Generator

CollapseImage	System.Drawing.Bitmap
ExpandImage	System.Drawing.Bitmap
Height	20
LineColor	Black
LineStyle	Dotted
LineWidth	1
NoExpandImage	System.Drawing.Bitmap
TransparentColor	Magenta
Width	19

Width
The width of each tree line image.

Folder name: TreeLineImages

Figure 14-16

From within this dialog, you can select the images used for the nodes that require an Expand, Collapse, or NoCollapse icon. You can also specify the color and style of the lines that connect the nodes. As you create your styles, a sample TreeView control output is displayed for you directly in the dialog box based on how your styles are to be applied. The final step is to choose the output of the files that this dialog will create. When you have completed this step, click OK. This generates a long list of new files to the folder that you specified in the dialog. By default, the ASP.NET TreeView Line Image Generator wants you to name the output folder `TreeLineImages`, but feel free to name it as you wish. If the folder does not exist in the project, you are prompted to allow Visual Studio to create the folder for you. After this is in place, the TreeView control can use your new images and styles if you set the `LineImagesFolder` property as follows:

```
<asp:TreeView ID="TreeView1" runat="server" ShowLines="True"
 DataSourceId="SiteMapDataSource1" LineImagesFolder="TreeViewLineImages">
```

The important properties are shown in bold. The `ShowLines` property must be set to `True`. After it is set, it uses the default settings displayed earlier, unless you have specified a location where it can retrieve custom images and styles using the `LineImagesFolder` property. As you can see, this simply points to the new folder, `TreeViewLineImages`, which contains all the new images and styles you created. Look in the folder — it is interesting to see what is output by the tool.

Working with the TreeView Control Programmatically

So far, with the TreeView control, you have learned how to work with the control declaratively. The great thing about ASP.NET is that you are not simply required to work with its components declaratively, but you can also manipulate these controls programmatically.

The TreeView control has an associated `TreeView` class that enables you to completely manage the TreeView control and how it functions from within your code. The next section looks at how to use some of the more common ways to control the `TreeView` programmatically.

Expanding and Collapsing Nodes Programmatically

One thing you can do with your TreeView control is to expand or collapse the nodes within the hierarchy programmatically. You can accomplish this by using either the `ExpandAll` or `CollapseAll` methods from the `TreeView` class. Listing 14-12 shows you one of the earlier TreeView controls that you used in Listing 14-6, but with a couple of buttons above it that you can now use to initiate the expanding and collapsing of the nodes.

Listing 14-12: Expanding and collapsing the nodes of the TreeView control programmatically

VB

```
<%@ Page Language="VB" %>

<script runat="server">
   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        TreeView1.ExpandAll()
   End Sub

   Protected Sub Button2_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        TreeView1.CollapseAll()
   End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>TreeView Control</title>
</head>
<body>
    <form id="Form1" runat="server">
        <p>
            <asp:Button ID="Button1" runat="server" Text="Expand Nodes"
             OnClick="Button1_Click" />
            <asp:Button ID="Button2" runat="server" Text="Collapse Nodes"
             OnClick="Button2_Click" />
            <br />
            <br />
        <asp:TreeView ID="TreeView1" runat="server"
         DataSourceId="SiteMapDataSource1">
```

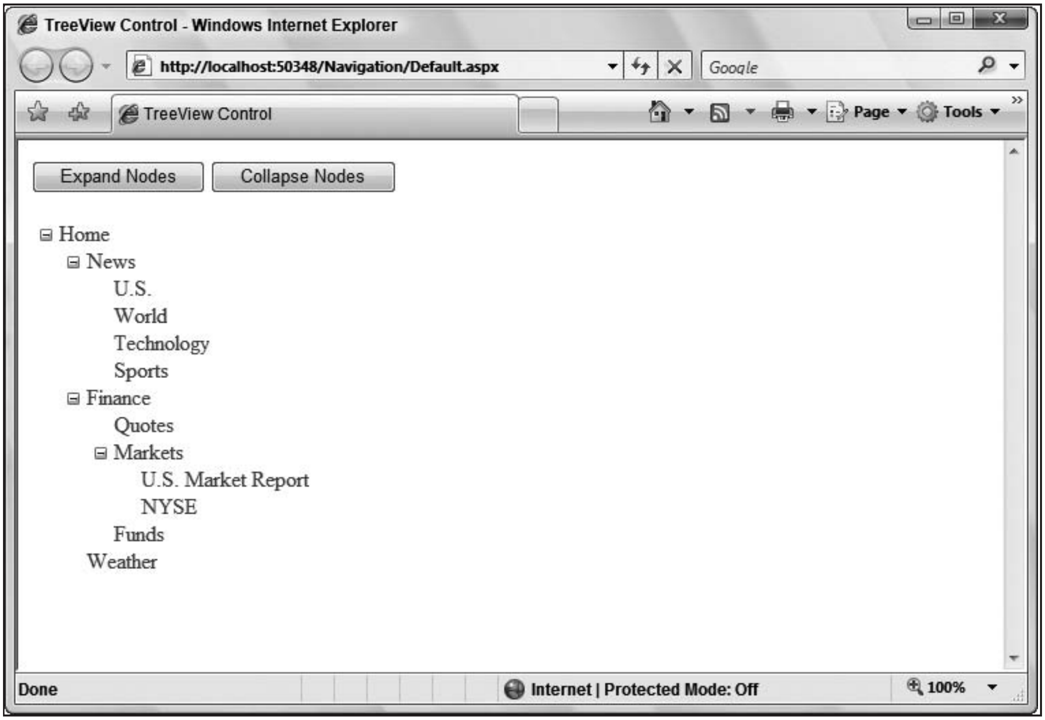
Continued

```
        </asp:TreeView>
        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" /></p>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, System.EventArgs e)
    {
        TreeView1.ExpandAll();
    }

    protected void Button2_Click(object sender, System.EventArgs e)
    {
        TreeView1.CollapseAll();
    }
</script>
```

Running this page gives you two buttons above your TreeView control. Clicking the first button invokes the `ExpandAll()` method and completely expands the entire list of nodes. Clicking the second button invokes the `CollapseAll()` method and completely collapses the list of nodes (see Figure 14-17).


TreeView Control - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
TreeView Control
Page
Tools
Expand Nodes
Collapse Nodes
Done
Internet | Protected Mode: Off
100%

```
⊟ Home
    ⊟ News
          U.S.
          World
          Technology
          Sports
    ⊟ Finance
          Quotes
        ⊟ Markets
              U.S. Market Report
              NYSE
          Funds
      Weather
```

Figure 14-17

The example shown in Listing 14-12 is nice, but it expands and collapses the nodes only on end-user actions (when the end user clicks the button). It would be even nicer if you could initiate this action programmatically.

You might want to simply place the `TreeView1.ExpandAll()` command within the `Page_Load` event, but if you try this, you see that it doesn't work. Instead, you use the `OnDataBound` attribute within the TreeView control:

```
<asp:TreeView ID="TreeView1" runat="server"
 DataSourceId="SiteMapDataSource1" OnDataBound="TreeView1_DataBound">
</asp:TreeView>
```

The value of this attribute points to a method in your code, as shown here:

VB

```
Protected Sub TreeView1_DataBound(ByVal sender As Object, _
  ByVal e As System.EventArgs)
   TreeView1.ExpandAll()
End Sub
```

C#

```
protected void TreeView1_DataBound(object sender, System.EventArgs e)
{
   TreeView1.ExpandAll();
}
```

Now when you run the page, notice that the TreeView control is completely expanded when the page is first loaded in the browser.

You can also expand specific nodes within the tree instead of just expanding the entire list. For this example, use the `TreeView1_DataBound` method you just created. Using the site map from Listing 14-1, change the `TreeView1_DataBound` method so that it appears as shown in Listing 14-13.

Listing 14-13: Expanding specific nodes programmatically

VB

```
Protected Sub TreeView1_DataBound(ByVal sender As Object, _
  ByVal e As System.EventArgs)
   TreeView1.CollapseAll()
   TreeView1.FindNode("Home").Expand()
   TreeView1.FindNode("Home/Finance").Expand()
   TreeView1.FindNode("Home/Finance/Markets").Expand()
End Sub
```

C#

```
protected void TreeView1_DataBound(object sender, System.EventArgs e)
{
   TreeView1.CollapseAll();
   TreeView1.FindNode("Home").Expand();
   TreeView1.FindNode("Home/Finance").Expand();
   TreeView1.FindNode("Home/Finance/Markets").Expand();
}
```

In this case, you use the `FindNode()` method and expand the node that is found. The `FindNode()` method takes a `String` value, which is the node and the path of the node that you want to reference. For instance, `TreeView1.FindNode("Home/Finance").Expand()` expands the `Finance` node. To find the node, it is important to specify the entire path from the root node to the node you want to work with (in this case, the `Finance` node). You separate the nodes within the site map path structure with a forward-slash between each of the nodes in the site map path.

Note that you had to expand each of the nodes individually until you got to the `Finance` node. If you simply used `TreeView1.FindNode("Home/Finance/Markets").Expand()` in the `TreeView1_DataBound()` method, the `Finance` node would indeed be expanded, but the parent nodes above it (the `Finance` and `Home` nodes) would not have been expanded and you wouldn't see the expanded `Markets` node when invoking the page. (Try it; it's interesting.)

Instead of using the `Expand` method, you can just as easily set the `Expanded` property to `True`, as shown in Listing 14-14.

Listing 14-14: Expanding nodes programmatically using the Expanded property

VB

```
Protected Sub TreeView1_DataBound(ByVal sender As Object, _
  ByVal e As System.EventArgs)
   TreeView1.CollapseAll()
   TreeView1.FindNode("Home").Expanded = True
   TreeView1.FindNode("Home/Finance").Expanded = True
   TreeView1.FindNode("Home/Finance/Markets").Expanded = True
End Sub
```

C#

```
protected void TreeView1_DataBound(object sender, System.EventArgs e)
{
   TreeView1.CollapseAll();
   TreeView1.FindNode("Home").Expanded = true;
   TreeView1.FindNode("Home/Finance").Expanded = true;
   TreeView1.FindNode("Home/Finance/Markets").Expanded = true;
}
```

Although you focus on the `Expand` method and the `Expanded` property here, you can just as easily programmatically collapse nodes using the `Collapse()` method. No `Collapsed` property really exists. Instead, you simply set the `Expanded` property to `False`.

Adding Nodes

Another interesting thing you can do with the TreeView control is to add nodes to the overall hierarchy programmatically. The TreeView control is made up of a collection of `TreeNode` objects. Therefore, as you see in previous examples, the `Finance` node is actually a `TreeNode` object that you can work with programmatically. It includes the capability to add other `TreeNode` objects.

A `TreeNode` object typically stores a `Text` and `Value` property. The `Text` property is what is displayed in the TreeView control for the end user. The `Value` property is an additional data item that you can use to associate with this particular `TreeNode` object. Another property that you can use (if your TreeView control is a list of navigational links) is the `NavigateUrl` property. Listing 14-15 demonstrates how to add nodes programmatically to the same site map from Listing 14-1 that you have been using.

Listing 14-15: Adding nodes programmatically to the TreeView control

VB

```
<%@ Page Language="VB" %>

<script runat="server">
   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        TreeView1.ExpandAll()
   End Sub

   Protected Sub Button2_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        TreeView1.CollapseAll()
   End Sub

   Protected Sub Button3_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim myNode As New TreeNode
        myNode.Text = TextBox1.Text
        myNode.NavigateUrl = TextBox2.Text
        TreeView1.FindNode("Home/Finance/Markets").ChildNodes.Add(myNode)
   End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>TreeView Control</title>
</head>
<body>
    <form id="Form1" runat="server">
        <p>
            <asp:Button ID="Button1" runat="server" Text="Expand Nodes"
             OnClick="Button1_Click" />
            <asp:Button ID="Button2" runat="server" Text="Collapse Nodes"
             OnClick="Button2_Click" /></p>
        <p>
            <strong>Text of new node:</strong>
            <asp:TextBox ID="TextBox1" runat="server">
            </asp:TextBox>
        </p>
        <p>
            <strong>Destination URL of new node:</strong>
            <asp:TextBox ID="TextBox2" runat="server">
            </asp:TextBox>
            <br />
            <br />
            <asp:Button ID="Button3" runat="server" Text="Add New Node"
             OnClick="Button3_Click" />
        </p>
        <p>
        <asp:TreeView ID="TreeView1" runat="server"
         DataSourceId="SiteMapDataSource1">
        </asp:TreeView></p>
        <p>
        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" /></p>
```

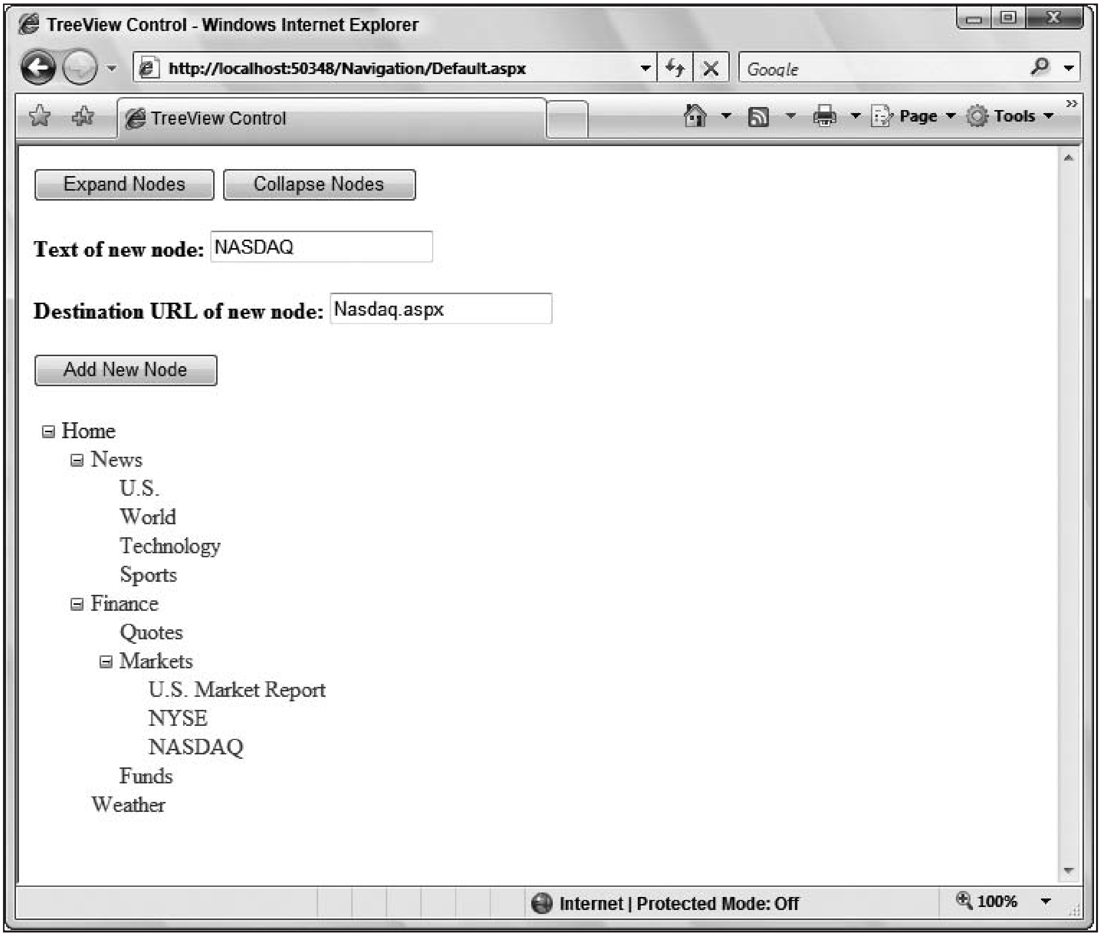
Continued

```
    </form>
</body>
</html>
```

C#

```
protected void Button3_Click(object sender, System.EventArgs e)
{
   TreeNode myNode = new TreeNode();
   myNode.Text = TextBox1.Text;
   myNode.NavigateUrl = TextBox2.Text;
   TreeView1.FindNode("Home/Finance/Markets").ChildNodes.Add(myNode);
}
```

This page contains two text boxes and a new Button control. The first text box is used to populate the `Text` property of the new node that is created. The second text box is used to populate the `NavigateUrl` property of the new node.


TreeView Control - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
TreeView Control
Page
Tools
Expand Nodes
Collapse Nodes

Text of new node: NASDAQ

Destination URL of new node: Nasdaq.aspx

Add New Node
Home
News
U.S.
World
Technology
Sports
Finance
Quotes
Markets
U.S. Market Report
NYSE
NASDAQ
Funds
Weather
Internet | Protected Mode: Off
100%

Figure 14-18

If you run the page, you can expand the entire hierarchy by clicking the `Expand Nodes` button. Then you can add additional child nodes to the `Markets` node. To add a new node programmatically, use the `FindNode` method as you did before to find the `Markets` node. When you find it, you can add additional child nodes by using the `ChildNodes.Add()` method and pass in a `TreeNode` object instance. Submitting `NASDAQ` in the first text box and `Nasdaq.aspx` in the second text box changes your TreeView control as illustrated in Figure 14-18.

After it is added, the node stays added even after the hierarchy tree is collapsed and re-opened. You can also add as many child nodes as you want to the `Markets` node. Note that, although you are changing nodes programmatically, this in no way alters the contents of the data source (the XML file, or the `.sitemap` file). These sources remain unchanged throughout the entire process.

Menu Server Control

One of the cooler navigation controls found in ASP.NET 3.5 is the Menu server control. This control is ideal for allowing the end user to navigate a larger hierarchy of options while utilizing very little browser real estate in the process. Figure 14-19 shows you what the menu control looks like when it is either completely collapsed or completely extended down one of the branches of the hierarchy.

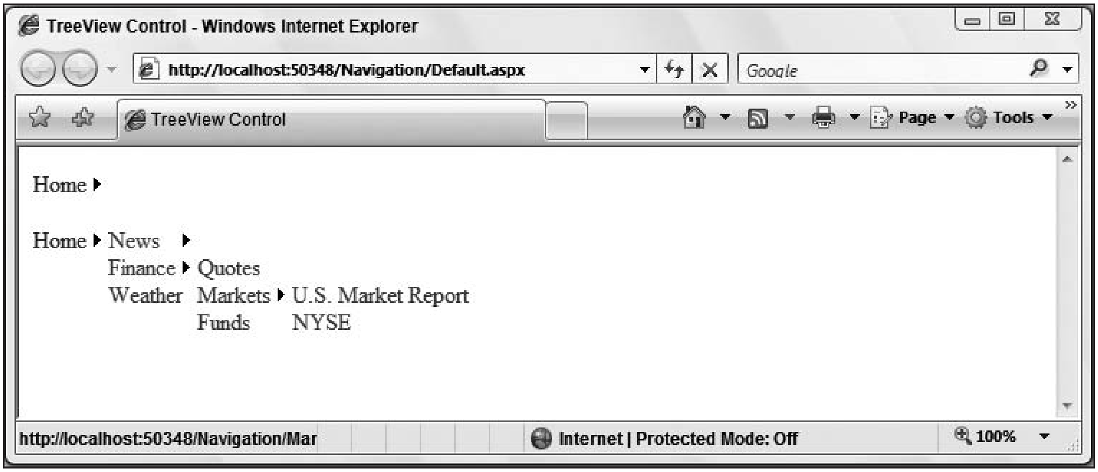

TreeView Control - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
TreeView Control
Page
Tools
Home ▸
Home ▸ News ▸
Finance ▸ Quotes
Weather Markets ▸ U.S. Market Report
Funds NYSE
http://localhost:50348/Navigation/Mar
Internet | Protected Mode: Off
100%

Figure 14-19

From here, you can see that the first Menu control displayed simply shows the Home link with a small arrow to the right of the display. The arrow means that more options are available that relate to this up-most link in the hierarchy. The second Menu control displayed shows what the default control looks like when the end user works down one of the branches provided by the site map.

The Menu control is an ideal control to use when you have lots of options — whether these options are selections the end user can make or navigation points provided by the application in which they are working. The Menu control can provide a multitude of options and consumes little space in the process.

Using the Menu control in your ASP.NET applications is rather simple. The Menu control works with a SiteMapDataSource control. You can drag and drop the SiteMapDataSource control and the Menu control onto the Visual Studio 2008 design surface and connect the two by using the Menu control's `DataSourceId` property. Alternatively, you can create and connect them directly in code. Listing 14-16 shows an example of the Menu control in its simplest form.

Listing 14-16: A simple use of the Menu control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Menu Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
        <asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1">
        </asp:Menu>
    </form>
</body>
</html>
```

From this example, you can see that I'm using a SiteMapDataSource control that automatically works with the application's `Web.sitemap` file. The only other item included is the Menu control, which uses the typical `ID` and `runat` attributes and the `DataSourceID` attribute to connect it with what is retrieved from the SiteMapDataSource control.

Although the default Menu control is pretty simple, you can highly customize how this control looks and works by redefining the properties of the control. The following sections look at some examples of how you can modify the appearance and change the behavior of this control.

Applying Different Styles to the Menu Control

By default, the Menu control is pretty plain. If you want to maintain this appearance, you can use what is provided or simply change the font sizes and styles to make it fit in with your site. You actually have quite a number of ways in which you can modify this control so that it appears unique and fits in with the rest of your site. Either you can customize this control's appearance yourself, or you can use one of the predefined styles that come with the control.

Using a Predefined Style

Visual Studio 2008 includes some predefined styles that you can use with the Menu control to quickly apply a look-and-feel to the displayed menu of items. Some of the provided styles include `Classic` and `Professional` and more. To apply one of these predefined styles, you work with the Menu control from the Design view of your page. Within the Design view, highlight the Menu control and expand the control's smart tag. From here, you see a list of options for working with this control. To change the look-and-feel of the control, click the Auto Format link and select one of the styles.

Performing this operation changes the code of your control by applying a set of style properties. For example, if you select the `Classic` option, you get the results shown in Listing 14-17.

Listing 14-17: Code changes when a style is applied to the Menu control

```
<asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1"
 BackColor="#B5C7DE" ForeColor="#284E98"
 Font-Names="Verdana" Font-Size="0.8em" StaticSubMenuIndent="10px"
 DynamicHorizontalOffset="2">
   <StaticSelectedStyle BackColor="#507CD1"></StaticSelectedStyle>
   <StaticMenuItemStyle HorizontalPadding="5"
    VerticalPadding="2"></StaticMenuItemStyle>
   <DynamicMenuStyle BackColor="#B5C7DE"></DynamicMenuStyle>
   <DynamicSelectedStyle BackColor="#507CD1"></DynamicSelectedStyle>
   <DynamicMenuItemStyle HorizontalPadding="5"
    VerticalPadding="2"></DynamicMenuItemStyle>
   <DynamicHoverStyle ForeColor="White" Font-Bold="True"
    BackColor="#284E98"></DynamicHoverStyle>
   <StaticHoverStyle ForeColor="White" Font-Bold="True"
    BackColor="#284E98"></StaticHoverStyle>
</asp:Menu>
```

You can see a lot of added styles that change the menu items that appear in the control. Figure 14-20 shows how this style selection appears in the browser.

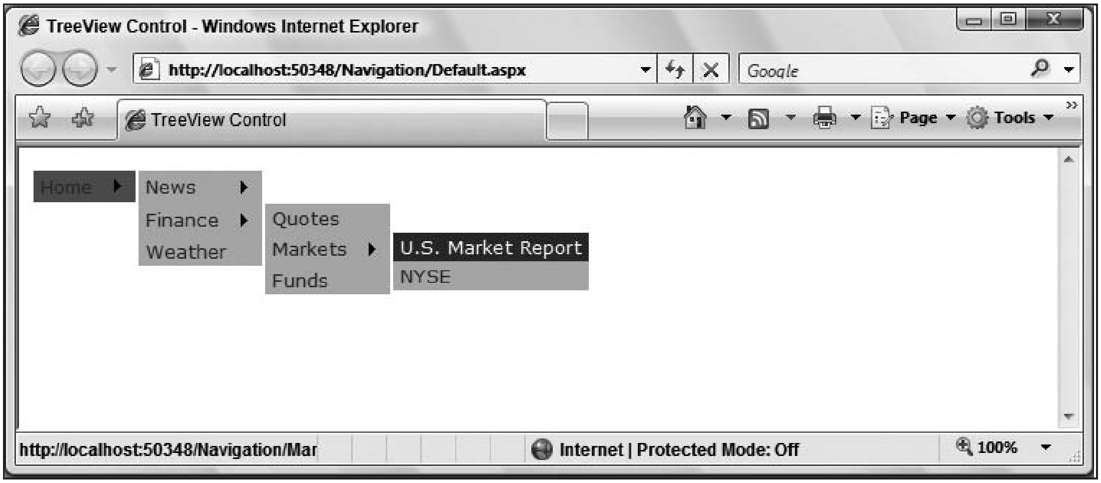

TreeView Control - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
TreeView Control
Page
Tools
Home
News
Finance
Weather
Quotes
Markets
Funds
U.S. Market Report
NYSE
http://localhost:50348/Navigation/Mar
Internet | Protected Mode: Off
100%

Figure 14-20

Changing the Style for Static Items

The Menu control considers items in the hierarchy to be either *static* or *dynamic*. Static items from this example would be the `Home` link that appears when the page is generated. Dynamic links are the items that appear dynamically when the user hovers the mouse over the `Home` link in the menu. It is possible to change the styles for both these types of nodes in the menu.

To apply a specific style to the static links that appear, you must add a static style element to the Menu control. The Menu control includes the following static style elements:

- ☐ `<StaticHoverStyle>`
- ☐ `<StaticItemTemplate>`
- ☐ `<StaticMenuItemStyle>`
- ☐ `<StaticMenuStyle>`
- ☐ `<StaticSelectedStyle>`

The important options from this list include the `<StaticHoverStyle>` and the `<StaticMenuItemStyle>` elements. The `<StaticHoverStyle>` is what you use to define the style of the static item in the menu when the end user hovers the mouse over the option. The `<StaticMenuItemStyle>` is what you use for the style of the static item when the end user is not hovering the mouse over the option.

Listing 14-18 illustrates adding a style that is applied when the end user hovers the mouse over static items.

Listing 14-18: Adding a hover style to static items in the menu control

```
<asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1">
   <StaticHoverStyle BackColor="DarkGray" BorderColor="Black" BorderStyle="Solid"
    BorderWidth="1"></StaticHoverStyle>
</asp:Menu>
```

This little example adds a background color and border to the static items in the menu when the end user hovers the mouse over the item. The result is shown in Figure 14-21.

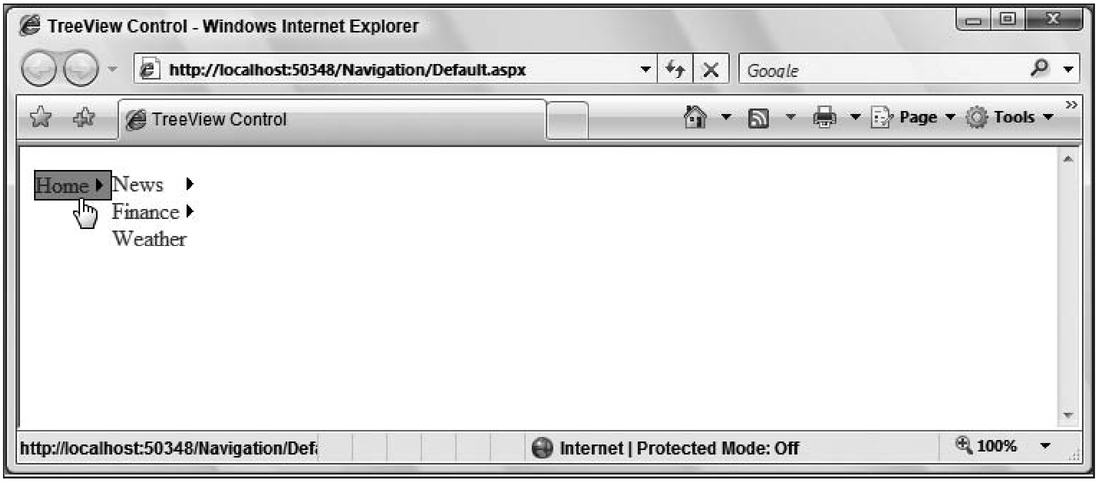

TreeView Control - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
TreeView Control
Page
Tools
Home
News
Finance
Weather
http://localhost:50348/Navigation/Def
Internet | Protected Mode: Off
100%

Figure 14-21

Adding Styles to Dynamic Items

Adding styles to the dynamic items of the menu control is just as easy as adding them to static items. The Menu control has a number of different elements for modifying the appearance of dynamic items, including the following:

- `<DynamicHoverStyle>`
- `<DynamicItemTemplate>`
- `<DynamicMenuItemStyle>`
- `<DynamicMenuStyle>`
- `<DynamicSelectedStyle>`

These elements change menu items the same way as the static versions of these elements, but they change only the items that dynamically pop-out from the static items. Listing 14-19 shows an example of applying the hover style to dynamic items.

Listing 14-19: Adding a hover style to dynamic items in the menu control

```
<asp:Menu ID="Menu1" runat="server" DataSourceID="Sitemapdatasource1">
   <StaticHoverStyle BackColor="DarkGray" BorderColor="Black" BorderStyle="Solid"
    BorderWidth="1"></StaticHoverStyle>
   <DynamicHoverStyle BackColor="DarkGray" BorderColor="Black" BorderStyle="Solid"
    BorderWidth="1"></DynamicHoverStyle>
</asp:Menu>
```

This code produces the results shown in Figure 14-22.

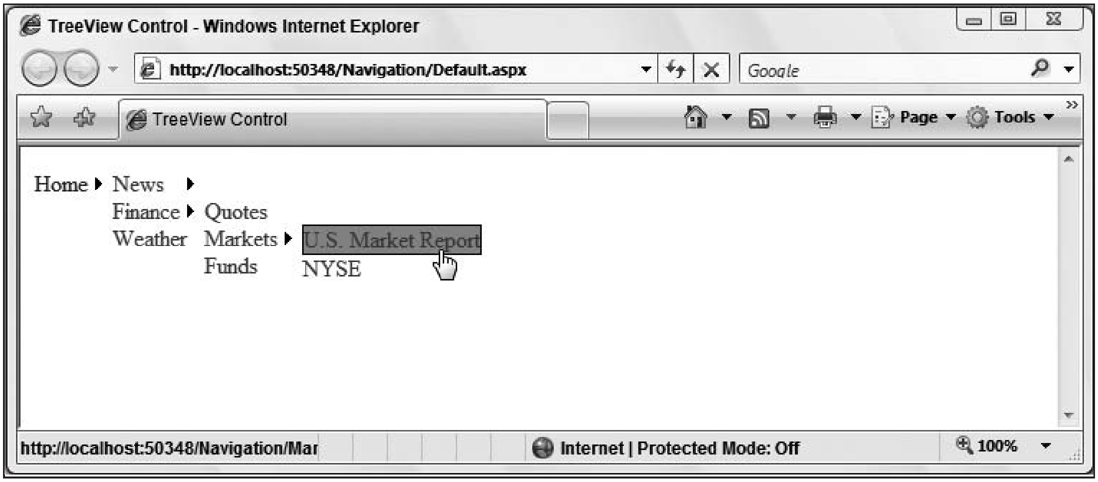

TreeView Control - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
TreeView Control
Page
Tools
Home
News
Finance
Weather
Quotes
Markets
Funds
U.S. Market Report
NYSE
http://localhost:50348/Navigation/Mar
Internet | Protected Mode: Off
100%

Figure 14-22

Changing the Layout of the Menu Items

By default, the dynamic menu items are displayed from left to right. This means that, as the items in the menu expand, they are continually displayed in a vertical fashion. You can actually control this behavior, but another option is available to you.

The other option is to have the first level of menu items appear directly below the first static item (horizontally). You change this behavior by using the `Orientation` attribute of the Menu control, as shown in Listing 14-20.

Listing 14-20: Forcing the menu items to use a horizontal orientation

```
<asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1"
 Orientation="Horizontal">
</asp:Menu>
```

This code produces the results shown in Figure 14-23.

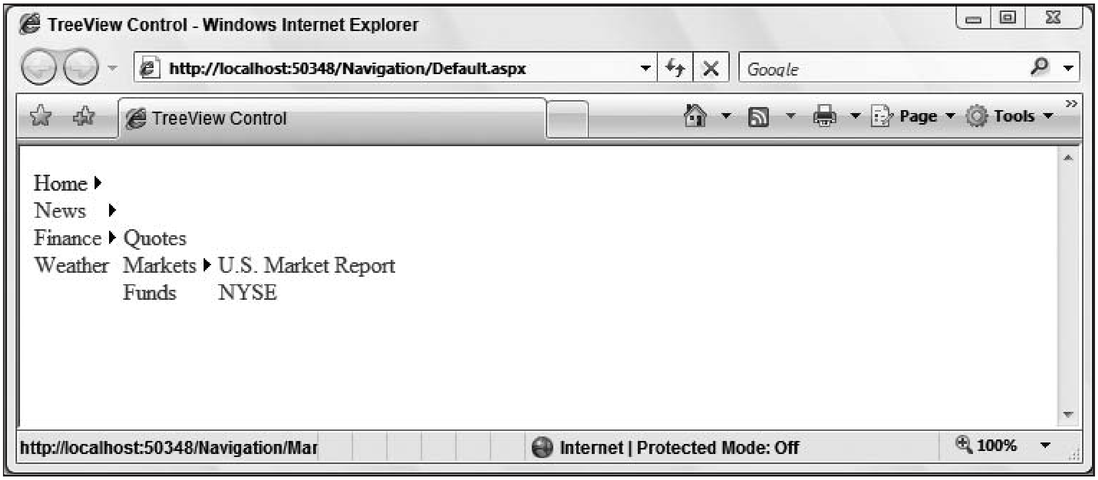

TreeView Control - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
TreeView Control
Page
Tools
Home
News
Finance
Weather
Quotes
Markets
Funds
U.S. Market Report
NYSE
http://localhost:50348/Navigation/Mar
Internet | Protected Mode: Off
100%

Figure 14-23

The `Orientation` attribute can take a value of `Horizontal` or `Vertical` only. The default value is `Vertical`.

Changing the Pop-Out Symbol

As the default, an arrow is used as the pop-out symbol for the menu items generated, whether they are static or dynamic. This is shown in Figure 14-24.

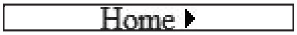


Figure 14-24

You are not forced to use this arrow symbol; in fact, you can change it to an image with relatively little work. Listing 14-21 shows how to accomplish this task.

Listing 14-21: Using custom images

```
<asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1"
 Orientation="Horizontal" DynamicPopOutImageUrl="myArrow.gif"
 StaticPopOutImageUrl="myArrow.gif">
</asp:Menu>
```

To change the pop-out symbol to an image of your choice, you use the `DynamicPopOutImageUrl` or `StaticPopOutImageUrl` properties. The `String` value these attributes take is simply the path of the image you want to use. Depending on the image used, it produces something similar to what you see in Figure 14-25.

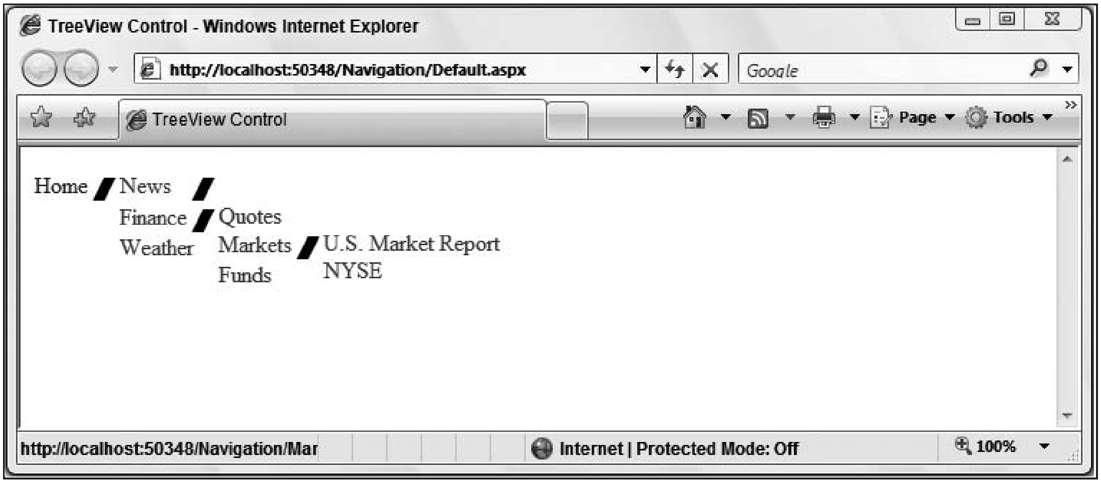

TreeView Control - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
TreeView Control
Page
Tools
Home
News
Finance
Weather
Quotes
Markets
Funds
U.S. Market Report
NYSE
http://localhost:50348/Navigation/Mar
Internet | Protected Mode: Off
100%

Figure 14-25

Separating Menu Items with Images

Another nice styling option of the Menu control is the capability to add a divider image to the menu items. You use the `StaticBottomSeparatorImageUrl`, `StaticTopSeparatorImageUrl`, `DynamicBottomSeparatorImageUrl`, and `DynamicTopSeparatorImageUrl` properties depending on where you want to place the separator image.

For example, if you wanted to place a divider image under only the dynamic menu items, you use the `DynamicBottomSeparatorImageUrl` property, as shown in Listing 14-22.

Listing 14-22: Applying divider images to dynamic items

```
<asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1"
 DynamicBottomSeparatorImageUrl="myDivider.gif">
</asp:Menu>
```

All the properties of the Menu control that define the image to use for the dividers take a `String` value that points to the location of the image. The result of Listing 14-22 is shown in Figure 14-26.

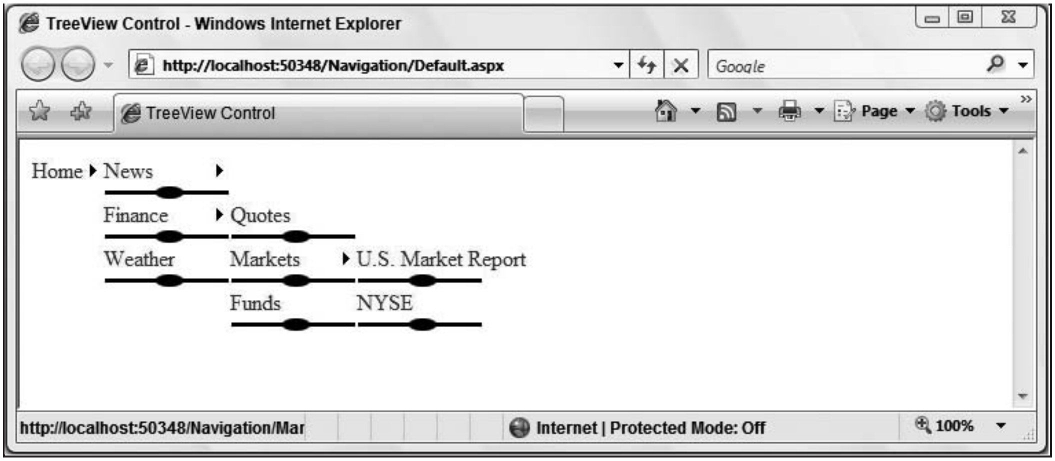

TreeView Control - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
TreeView Control
Page
Tools
Home
News
Finance
Quotes
Weather
Markets
U.S. Market Report
Funds
NYSE
http://localhost:50348/Navigation/Mar
Internet | Protected Mode: Off
100%

Figure 14-26

Menu Events

The Menu control exposes events such as the following:

- `DataBinding`
- `DataBound`
- `Disposed`
- `Init`
- `Load`
- `MenuItemClick`
- `MenuItemDataBound`
- `PreRender`
- `Unload`

One nice event to be aware of is the `MenuItemClick` event. This event, shown in Listing 14-23, enables you to take some action when the end user clicks one of the available menu items.

Listing 14-23: Using the MenuItemClick event

VB

```
Protected Sub Menu1_MenuItemClick(ByVal sender As Object, _
   ByVal e As System.Web.UI.WebControls.MenuEventArgs)

   ' Code for event here

End Sub
```

C#

```
protected void Menu1_MenuItemClick(object sender, MenuEventArgs e)
{

   // Code for event here

}
```

This delegate uses the `MenuEventArgs` data class and provides you access to the text and value of the item selected from the menu.

Binding the Menu Control to an XML File

Just as with the TreeView control, it is possible to bind the Menu control to items that come from other data source controls provided with ASP.NET 3.5. Although most developers are likely to use the Menu control to enable end users to navigate to URL destinations, you can also use the Menu control to enable users to make selections.

As an example, take the previous XML file, `Hardware.xml`, which was used with the TreeView control from Listing 14-8 earlier in the chapter. For this example, the Menu control works with an XmlDataSource control. When the end user makes a selection from the menu, you populate a Listbox on the page with the items selected. The code for this is shown in Listing 14-24.

Listing 14-24: Using the Menu control with an XML file

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Menu1_MenuItemClick(ByVal sender As Object, _
       ByVal e As System.Web.UI.WebControls.MenuEventArgs)

        Listbox1.Items.Add(e.Item.Parent.Value & " : " & e.Item.Value)
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Menu Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Menu ID="Menu1" runat="server" DataSourceID="XmlDataSource1"
         OnMenuItemClick="Menu1_MenuItemClick">
            <DataBindings>
                <asp:MenuItemBinding DataMember="Item"
                 TextField="Category"></asp:MenuItemBinding>
                <asp:MenuItemBinding DataMember="Option"
                 TextField="Choice"></asp:MenuItemBinding>
            </DataBindings>
```

Continued

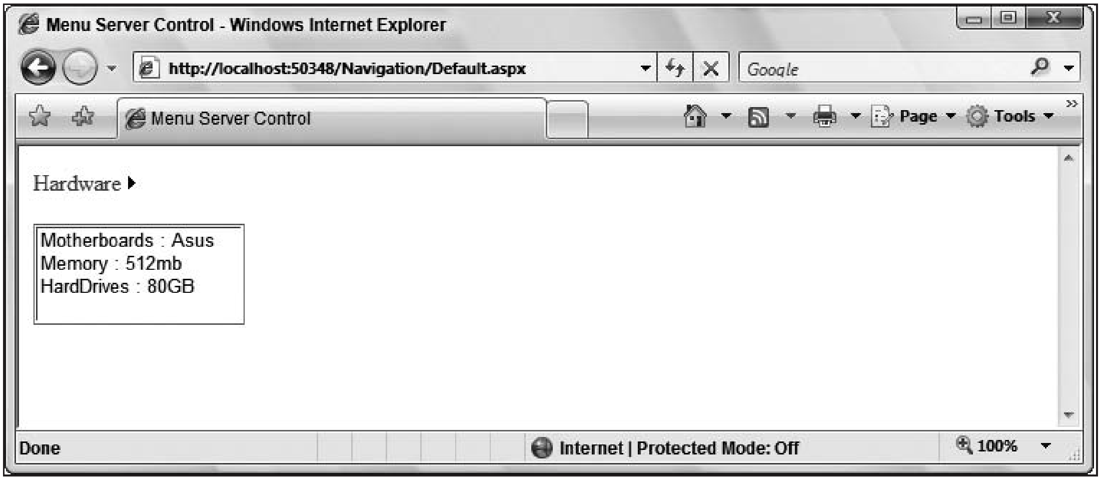
```
            </asp:Menu>
            <p>
            <asp:ListBox ID="Listbox1" runat="server">
            </asp:ListBox></p>
            <asp:xmldatasource ID="XmlDataSource1" runat="server"
             datafile="Hardware.xml" />
        </form>
    </body>
    </html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
   protected void Menu1_MenuItemClick(object sender, MenuEventArgs e)
   {
      Listbox1.Items.Add(e.Item.Parent.Value + " : " + e.Item.Value);
   }
</script>
```

From this example, you can see that instead of using the `<asp:TreeNodeBinding>` elements, as we did with the TreeView control, the Menu control uses the `<asp:MenuItemBinding>` elements to make connections to items listed in the XML file, `Hardware.xml`. In addition, the root element of the Menu control, the `<asp:Menu>` element, now includes the `OnMenuItemClick` attribute, which points to the event delegate `Menu1_MenuItemClick`.

The `Menu1_MenuItemClick` delegate includes the data class `MenuEventArgs`, which enables you to get at both the values of the child and parent elements selected. For this example, both are used and then populated into the Listbox control, as illustrated in Figure 14-27.


Menu Server Control - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
Menu Server Control
Page
Tools
Hardware
Done
Internet | Protected Mode: Off
100%

Motherboards : Asus
Memory : 512mb
HardDrives : 80GB

Figure 14-27

SiteMap Data Provider

A series of data providers in the form of DataSource controls are available in ASP.NET 3.5. One of these DataSource controls now at your disposal, which you looked at earlier in the chapter, is the SiteMapDataSource control. This DataSource control was developed to work with site maps and the controls that can bind to them.

Some controls do not need a SiteMapDataSource control in order to bind to the application's site map (which is typically stored in the `Web.sitemap` file). Earlier in the chapter, you saw this in action when using the SiteMapPath control. This control was able to work with the `Web.sitemap` file directly — without the need for this data provider.

Certain navigation controls, however, such as the TreeView control and the DropDownList control, require an intermediary SiteMapDataSource control to retrieve the site navigation information.

The SiteMapDataSource control is simple to use as demonstrated throughout this chapter. The SiteMapDataSource control in its simplest form is illustrated here:

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
```

In this form, the SiteMapDataSource control simply grabs the info as a tree hierarchy (as consistently demonstrated so far). Be aware that a number of properties do change how the data is displayed in any control that binds to the data output.

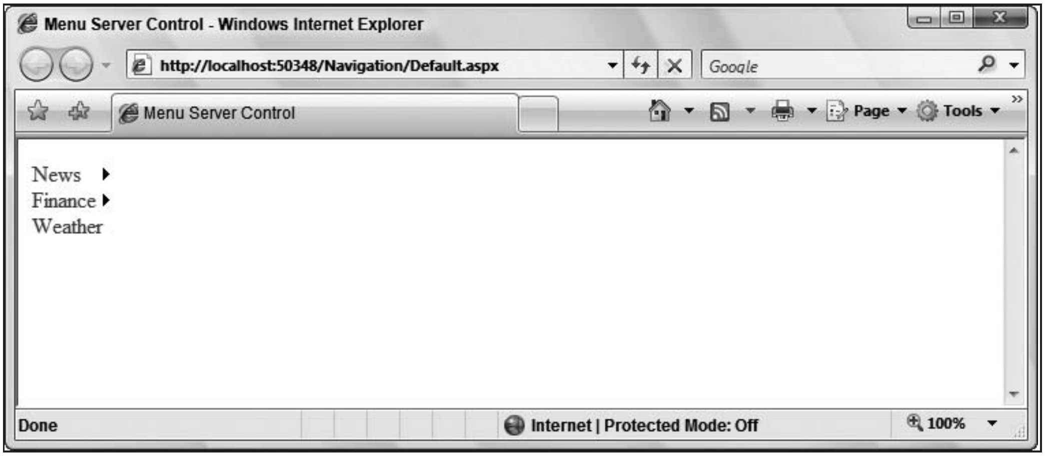
ShowStartingNode

The `ShowStartingNode` property determines whether the root node of the `.sitemap` file is retrieved with the retrieved collection of node objects. This property takes a `Boolean` value and is set to `True` by default. If you are working with the `Web.sitemap` file shown in Listing 14-1, you construct your SiteMapDataSource control as shown in Listing 14-25 to remove the root node from the collection.

Listing 14-25: Removing the root node from the retrieved node collection

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Menu Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server"
         ShowStartingNode="False" />
        <asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1">
        </asp:Menu>
    </form>
</body>
</html>
```

This code produces a menu like the one shown in Figure 14-28.


Menu Server Control - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
Menu Server Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

News
Finance
Weather

Figure 14-28

From this screenshot, you can see that indeed the root node has been removed, and the menu shown starts by using all the child nodes of the root node.

StartFromCurrentNode

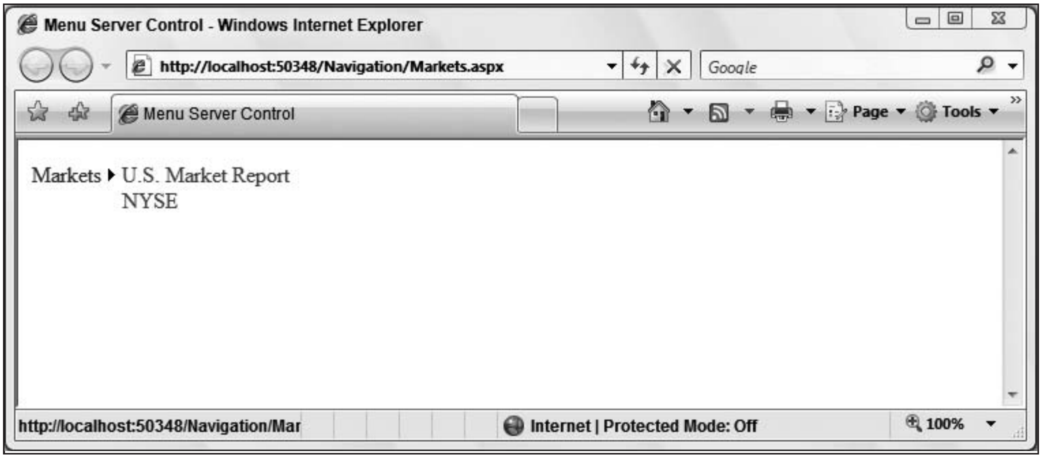
The `StartFromCurrentNode` property causes the SiteMapDataProvider to retrieve only a node collection that starts from the current node of the page being viewed. By default, this is set to `False`, meaning that the SiteMapDataProvider always retrieves all the available nodes (from the root node to the current node).

For an example of this, use the `.sitemap` file from Listing 14-1 and create a page called `Markets.aspx`. This page in the hierarchy of the node collection is a child node of the Finance node, as well as having two child nodes itself: U.S. Market Report and NYSE. An example of setting the `StartFromCurrentNode` property to `True` is shown in Listing 14-26.

Listing 14-26: The Markets.aspx page using the StartFromCurrentNode property

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Menu Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server"
         StartFromCurrentNode="True" />
        <asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1">
        </asp:Menu>
    </form>
</body>
</html>
```

This simple property addition produces the result shown in Figure 14-29.


Menu Server Control - Windows Internet Explorer
http://localhost:50348/Navigation/Markets.aspx
Google
Menu Server Control
Page
Tools
http://localhost:50348/Navigation/Mar
Internet | Protected Mode: Off
100%

Markets ▸ U.S. Market Report
NYSE

Figure 14-29

StartingNodeOffset

The `StartingNodeOffset` property takes an `Integer` value that determines the starting point of the hierarchy collection. Be default, this property is set to `0`, meaning that the node collection retrieved by the SiteMapDataSource control starts at the root node. Any other value provides the offset from the root node and, in turn, makes this the new starting point. From the example provided in Listing 14-1, you know that the collection starts with the Home page found at `Default.aspx`, a page that you have seen in numerous examples in this chapter.

If you set this property's value to `1`, the starting point of the collection is one space off the default starting point (the Home page starting at `Default.aspx`). For example, if the page using the SiteMapDataSource control is the `MarketsUS.aspx` page, the node collection starts with the Finance page (`Finance.aspx`).

```
Home   Offset 0
   News  Offset 1
      U.S.   Offset 2
      World   Offset 2
      Technology   Offset 2
      Sports   Offset 2
   Finance   Offset 1
      Quotes   Offset 2
      Markets   Offset 2
         U.S. Market Report   Offset 3
         NYSE   Offset 3
      Funds   Offset 2
   Weather   Offset 1
```

From this hierarchy, you can see how much each node is offset from the root node. Therefore, if you set the `StartingNodeOffset` property to `1` and you are browsing on the U.S. Market Report page, you can see that the node collection starts with the Finance page (`Finance.aspx`) and the other child nodes of the root node (News and Weather) are not represented in the node collection as the `Finance.aspx` page is on the direct hierarchical path of the requested page.

StartingNodeUrl

The `StartingNodeUrl` property enables you to specify the page found in the `.sitemap` file from which the node collection should start. By default, the value of this property is empty; but when set to something such as `Finance.aspx`,, the collection starts with the Finance page as the root node of the node collection. Listing 14-27 shows an example of using the `StartingNodeUrl` property.

Listing 14-27: Using the StartingNodeUrl property

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Menu Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server"
         StartingNodeUrl="Finance.aspx" />
        <asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1">
        </asp:Menu>
    </form>
</body>
</html>
```

When the `StartingNodeUrl` property value is encountered, the value is compared against the `url` attributes in the `Web.sitemap` file. When a match is found, the matched page is the one used as the root node in the node collection retrieved by the SiteMapDataSource control.

SiteMap API

The `SiteMap` class is an in-memory representation of the site's navigation structure. This is a great class for programmatically working around the hierarchical structure of your site. The `SiteMap` class comes with a couple of objects that make working with the navigation structure easy. These objects (or public properties) are described in the following table.

Properties	Description
`CurrentNode`	Retrieves a `SiteMapNode` object for the current page
`RootNode`	Retrieves a `SiteMapNode` object that starts from the root node and the rest of the site's navigation structure
`Provider`	Retrieves the default `SiteMapProvider` for the current site map
`Providers`	Retrieves a collection of available, named `SiteMapProvider` objects

Listing 14-28 shows an example of working with some `SiteMap` objects by demonstrating how to use the `CurrentNode` object from the `Markets.aspx` page.

Listing 14-28: Working with the CurrentNode object

VB

```
<%@ Page Language="VB" %>

<script runat="server" language="vb">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = SiteMap.CurrentNode.Description & "<br>" & _
            SiteMap.CurrentNode.HasChildNodes & "<br>" & _
            SiteMap.CurrentNode.NextSibling.ToString() & "<br>" & _
            SiteMap.CurrentNode.ParentNode.ToString() & "<br>" & _
            SiteMap.CurrentNode.PreviousSibling.ToString() & "<br>" & _
            SiteMap.CurrentNode.RootNode.ToString() & "<br>" & _
            SiteMap.CurrentNode.Title & "<br>" & _
            SiteMap.CurrentNode.Url
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>SiteMapDataSource</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Label ID="Label1" runat="server"></asp:Label>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
        Label1.Text = SiteMap.CurrentNode.Description + "<br>" +
            SiteMap.CurrentNode.HasChildNodes + "<br>" +
            SiteMap.CurrentNode.NextSibling.ToString() + "<br>" +
            SiteMap.CurrentNode.ParentNode.ToString() + "<br>" +
            SiteMap.CurrentNode.PreviousSibling.ToString() + "<br>" +
            SiteMap.CurrentNode.RootNode.ToString() + "<br>" +
            SiteMap.CurrentNode.Title + "<br>" +
            SiteMap.CurrentNode.Url;
    }
</script>
```

As you can see from this little bit of code, by using the `SiteMap` class and the `CurrentNode` object you can work with a plethora of information regarding the current page. Running this page, you get the following results printed to the screen:

```
The Latest Market Information
True
Funds
Finance
```

```
Quotes
Home
Markets
/SiteNavigation/Markets.aspx
```

Using the `CurrentNode` property, you can actually create your own style of the SiteMapPath control, as illustrated in Listing 14-29.

Listing 14-29: Creating a custom navigation display using the CurrentNode property

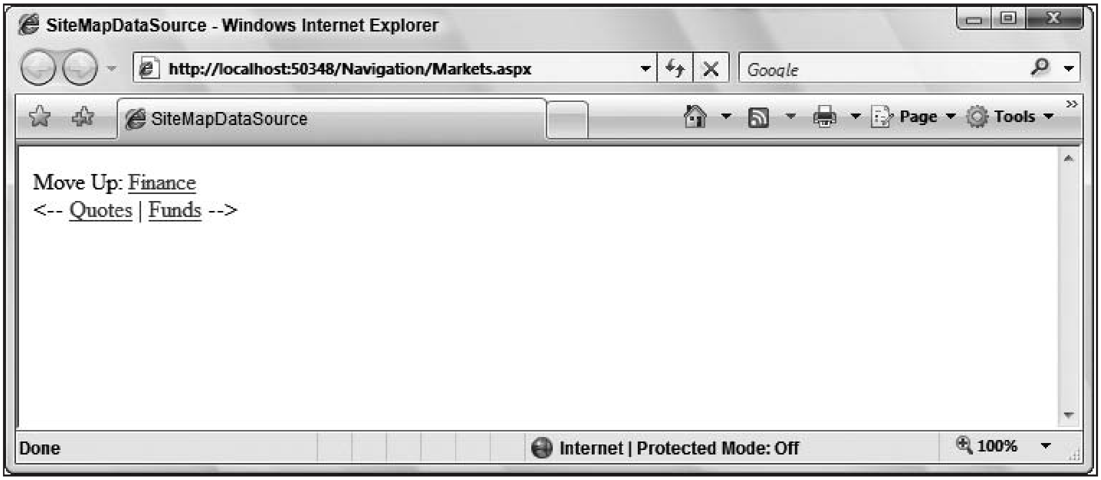
VB

```
<%@ Page Language="VB" %>
<script runat="server" language="vb">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Hyperlink1.Text = SiteMap.CurrentNode.ParentNode.ToString()
        Hyperlink1.NavigateUrl = SiteMap.CurrentNode.ParentNode.Url

        Hyperlink2.Text = SiteMap.CurrentNode.PreviousSibling.ToString()
        Hyperlink2.NavigateUrl = SiteMap.CurrentNode.PreviousSibling.Url

        Hyperlink3.Text = SiteMap.CurrentNode.NextSibling.ToString()
        Hyperlink3.NavigateUrl = SiteMap.CurrentNode.NextSibling.Url
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>SiteMapDataSource</title>
</head>
<body>
    <form id="form1" runat="server">
        Move Up:
        <asp:Hyperlink ID="Hyperlink1" runat="server"></asp:Hyperlink><br />
        <-- <asp:Hyperlink ID="Hyperlink2" runat="server"></asp:Hyperlink> |
        <asp:Hyperlink ID="Hyperlink3" runat="server"></asp:Hyperlink> -->
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
        Hyperlink1.Text = SiteMap.CurrentNode.ParentNode.ToString();
        Hyperlink1.NavigateUrl = SiteMap.CurrentNode.ParentNode.Url;

        Hyperlink2.Text = SiteMap.CurrentNode.PreviousSibling.ToString();
        Hyperlink2.NavigateUrl = SiteMap.CurrentNode.PreviousSibling.Url;

        Hyperlink3.Text = SiteMap.CurrentNode.NextSibling.ToString();
        Hyperlink3.NavigateUrl = SiteMap.CurrentNode.NextSibling.Url;
    }
</script>
```

When run, this page gives you your own custom navigation structure, as shown in Figure 14-30.


SiteMapDataSource - Windows Internet Explorer
http://localhost:50348/Navigation/Markets.aspx
Google
SiteMapDataSource
Page
Tools
Done
Internet | Protected Mode: Off
100%

Move Up: Finance
<-- Quotes | Funds -->

Figure 14-30

URL Mapping

The URLs used by Web pages can sometimes get rather complex as your application grows and grows. Sometimes, you could be presenting Web pages that change their content based on querystrings that are provided via the URL, such as:

```
http://www.asp.net/forums/view.aspx?forumid=12&categoryid=6
```

In other cases, your Web page might be so deep within a hierarchy of folders that the URL has become rather cumbersome for an end user to type or remember when they want to pull up the page later in their browser. There are also moments when you want a collection of pages to look like they are the same page or a single destination.

In cases such as these, you can take advantage of a ASP.NET feature called *URL mapping*. URL mapping enables you to map complex URLs to simpler ones. You accomplish this through settings you apply in the `web.config` file using the `<urlMappings>` element (see Listing 14-30).

Listing 14-30: Mapping URLs using the <urlMappings> element

```
<configuration>

   <system.web>

      <urlMappings>
         <add url="~/Content.aspx" mappedUrl="~/SystemNews.aspx?categoryid=5" />
      </urlMappings>

   </system.web>
</configuration>
```

In this example, we provide a fake URL — `Content.aspx` — that is mapped to a more complicated URL: `SystemNews.aspx?categoryid=5`. With this construction in place, when the end user types URL `Content.aspx`, the application knows to invoke the more complicated URL `SystemNews.aspx?categoryid=5` page. This takes place without the URL even being changed in the browser. Even after the page has completely loaded, the browser will still show the `Content.aspx` page as the destination — thereby tricking the end user in a sense.

It is important to note that in this situation, the end user is routed to `SystemNews.aspx?categoryid=5` no matter what — *even if a* `Content.aspx` *page exists*! Therefore, it is important to map to pages that are not actually contained within your application.

Sitemap Localization

The improved resource files (`.resx`) are a great way to localize ASP.NET applications. This localization of Web applications using ASP.NET is covered in Chapter 31 of this book. However, this introduction focused on applying localization features to the pages of your applications; we didn't demonstrate how to take this localization capability further by applying it to items such as the `Web.sitemap` file.

Structuring the Web.sitemap File for Localization

Just as it is possible to apply localization instructions to the pages of your ASP.NET Web applications, you can also use the same framework to accomplish your localization tasks in the `Web.sitemap` file. To show you this in action, Listing 14-31 constructs a `Web.sitemap` file somewhat similar to the one presented in Listing 14-1, but much simpler.

Listing 14-31: Creating a basic .sitemap file for localization

```
<?xml version="1.0" encoding="utf-8" ?>

<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0"
 enableLocalization="true">
 <siteMapNode url="Default.aspx" resourceKey="Home">
        <siteMapNode url="News.aspx" resourceKey="News">
               <siteMapNode url="News.aspx?cat=us" resourceKey="NewsUS" />
               <siteMapNode url="News.aspx?cat=world" resourceKey="NewsWorld" />
               <siteMapNode url="News.aspx?cat=tech" resourceKey="NewsTech" />
               <siteMapNode url="News.aspx?cat=sport" resourceKey="NewsSport" />
        </siteMapNode>
 </siteMapNode>
</siteMap>
```

Looking at Listing 14-31, you can see that we have a rather simple `Web.sitemap` file. To enable the localization capability from the `Web.sitemap` file, you have to turn this capability on by using the `enableLocalization` attribute in the `<siteMap>` element and setting it to `true`. Once enabled, you can then define each of the navigation nodes as you would normally, using the `<siteMapNode>` element. In this case, however, because you are going to define the contents of these navigation pieces (most notably the `title` and `description` attributes) in various `.resx` files, there is no need to repeatedly define these items in this file. That means you need to define only the `url` attribute for this example. It is important to note, however, that you could also define this attribute through your `.resx` files, thereby forwarding end users to different pages depending on their defined culture settings.

The next attribute to note is the `resourceKey` attribute used in the `<siteMapNode>` elements. This is the key that is used and defined in the various `.resx` files you will implement. Take the following `<siteMapNode>` element as an example:

```
<siteMapNode url="News.aspx" resourceKey="News">
   ...
</siteMapNode>
```

In this case, the value of the `resourceKey` (and the key that will be used in the `.resx` file) is `News`. This means that you are then able to define the values of the `title` and `description` attributes in the `.resx` file using the following syntax:

```
News.Title
News.Description
```

Now that the `Web.sitemap` is in place, the next step is to make some minor modifications to the `Web.config` file, as shown next.

Making Modifications to the Web.config File

Now that the `Web.sitemap` file is in place and ready, the next step is to provide some minor additions to the `Web.config` file. In order for your Web application to make an automatic detection of the culture of the users visiting the various pages you are providing, you need to set the `Culture` and `UICulture` settings in the `@Page` directive, or set these attributes for automatic detection in the `<globalization>` element of the `Web.config` file.

When you are working with navigation and the `Web.sitemap` file, as we are, it is actually best to make this change in the `Web.config` file so that it automatically takes effect on each and every page in your application. This makes it much simpler because you won't have to make these additions yourself to each and every page.

To make these changes, open your `Web.config` file and add a `<globalization>` element, as shown in Listing 14-32.

Listing 14-32: Adding culture detection to the Web.config file

```
<configuration>
   <system.web>

      <globalization culture="auto" uiCulture="auto" />

   </system.web>
</configuration>
```

For the auto-detection capabilities to occur, you simply need to set the `culture` and `uiCulture` attributes to `auto`. You could have also defined the values as `auto:en-US`, which means that the automatic culture detection capabilities should occur, but if the culture defined is not found in the various resource files, then use `en-US` (American English) as the default culture. However, because we are going to define a default `Web.sitemap` set of values, there really is no need for you to bring forward this construction.

Next, you need to create the assembly resources files that define the values used by the `Web.sitemap` file.

Creating Assembly Resource (.resx) Files

To create a set of assembly resource files that you will use with the `Web.sitemap` file, create a folder in your project called `App_GlobalResources`. If you are using Visual Studio 2008 or Visual Web Developer, you can add this folder by right-clicking on the project and selecting Add Folder ➪ App_-GlobalResources.

After the folder is in place, the next step is to add two assembly resource files to this folder. Name the first file `Web.sitemap.resx` and the second one `Web.sitemap.fi.resx`. Your goal with these two files is to have a default set of values for the `Web.sitemap` file that will be defined in the `Web.sitemap.resx` file, and a version of these values that has been translated to the Finnish language and is contained in the `Web.sitemap.fi.resx` file.

The `fi` value used in the name will be the file used by individuals who have their preferred language set to `fi-FI`. Other variations of these constructions are shown in the following table.

.resx File	Culture Served
`Web.sitemap.resx`	The default values used when the end user's culture cannot be identified through another `.resx` file
`Web.sitemap.en.resx`	The resource file used for all `en` (English) users
`Web.sitemap.en-gb.resx`	The resource file used for the English speakers of Great Britain
`Web.sitemap.fr-ca.resx`	The resource file used for the French speakers of Canada
`Web.sitemap.ru.resx`	The resource file used for Russian speakers

Now that the `Web.sitemap.resx` and `Web.sitemap.fi.resx` files are in place, the next step is to fill these files with values. To accomplish this task, you use the keys defined earlier directly in the `Web.sitemap` file. Figure 14-31 shows the result of this exercise.

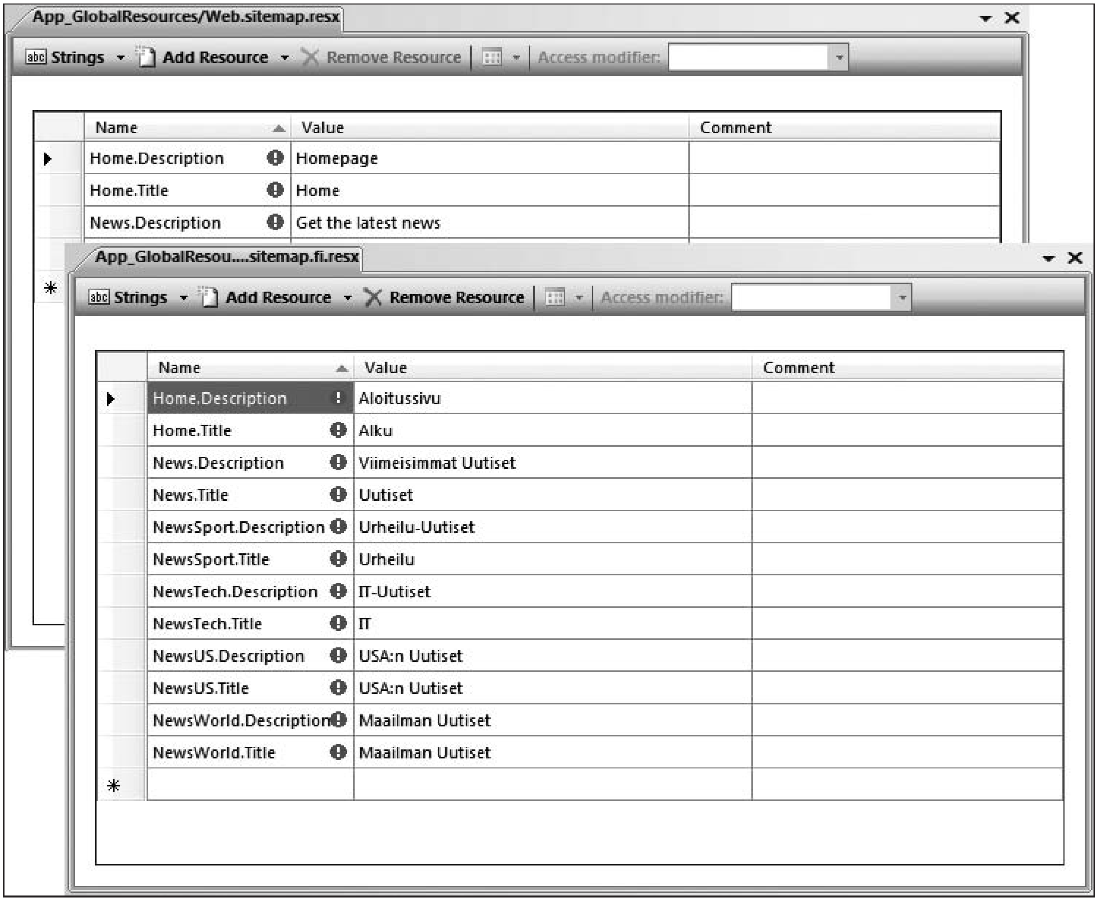
Although the IDE states that these are not valid identifiers, the application still works with this model. After you have the files in place, you can test how this localization endeavor works, as shown in the following section.

Testing the Results

Create a page in your application and place a TreeView server control on the page. In addition to the TreeView control, you also have to include a SiteMapDataSource control to work with the `Web.sitemap` file you created. Be sure to tie the two controls together by giving the TreeView control the attribute `DataSourceID="SiteMapDataSource1"`, as demonstrated earlier in this chapter.

If you have your language preference in Microsoft's Internet Explorer set to `en-us` (American English), you will see the results shown in Figure 14-32.

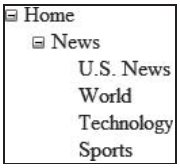
When you pull up the page in the browser, the culture of the request is checked. Because the only finely grained preference defined in the example is for users using the culture of `fi` (Finnish), the default `Web.sitemap.resx` is used instead. Because of this, the `Web.sitemap.resx` file is used to populate the values of the TreeView control, as shown in Figure 14-32. If the requestor has a culture setting of `fi`, however, he gets an entirely different set of results.


App_GlobalResources/Web.sitemap.resx
Strings
Add Resource
Remove Resource
Access modifier:
App_GlobalResou....sitemap.fi.resx
Strings
Add Resource
Remove Resource
Access modifier:

Name	Value	Comment
Home.Description	Homepage	
Home.Title	Home	
News.Description	Get the latest news	

Name	Value	Comment
Home.Description	Aloitussivu	
Home.Title	Alku	
News.Description	Viimeisimmat Uutiset	
News.Title	Uutiset	
NewsSport.Description	Urheilu-Uutiset	
NewsSport.Title	Urheilu	
NewsTech.Description	IT-Uutiset	
NewsTech.Title	IT	
NewsUS.Description	USA:n Uutiset	
NewsUS.Title	USA:n Uutiset	
NewsWorld.Description	Maailman Uutiset	
NewsWorld.Title	Maailman Uutiset	

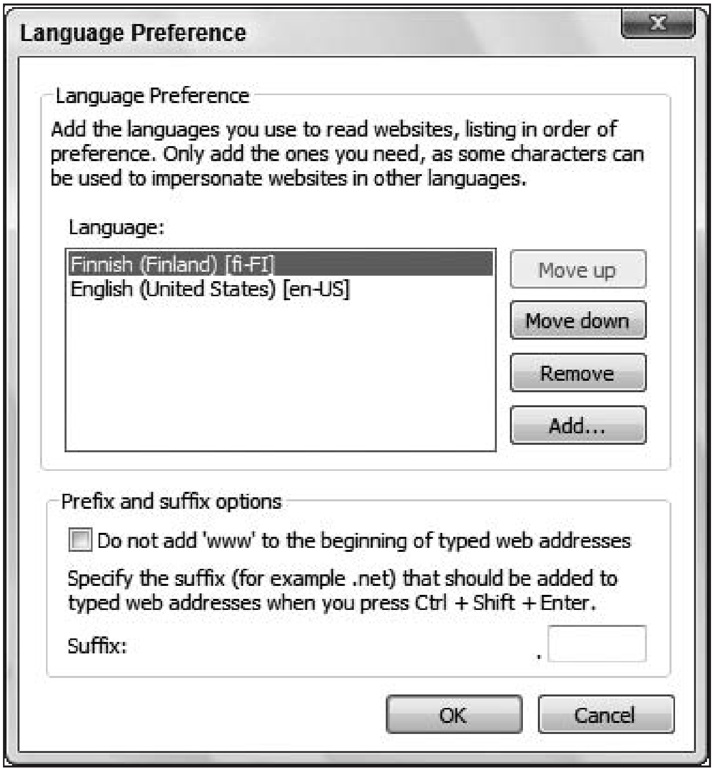
Figure 14-31



- Home
  - News
    - U.S. News
    - World
    - Technology
    - Sports

Figure 14-32

To test this out, change the preferred language used in IE by selecting Tools ➪ Internet Options in IE. On the first tab (General), click the Languages button at the bottom of the dialog. You are presented with the Language Preferences dialog. Click the Add button and add the Finnish language setting to the list of options. The final step is to use the Move Up button to move the Finnish choice to the top of the list. In the end, you should see something similar to what is shown in Figure 14-33.



Language Preference

Add the languages you use to read websites, listing in order of preference. Only add the ones you need, as some characters can be used to impersonate websites in other languages.

Language:

Finnish (Finland) [fi-FI]
English (United States) [en-US]

Prefix and suffix options

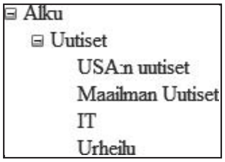
☐ Do not add 'www' to the beginning of typed web addresses

Specify the suffix (for example .net) that should be added to typed web addresses when you press Ctrl + Shift + Enter.

Suffix: .

Figure 14-33

With this setting in place, running the page with the TreeView control gives you the result shown in Figure 14-34.



```
⊟ Alku
    ⊟ Uutiset
          USA:n uutiset
          Maailman Uutiset
          IT
          Urheilu
```

Figure 14-34

Now, when the page is requested, the culture is set to `fi` and correlates to the `Web.sitemap.fi.resx` file instead of to the default `Web.sitemap.resx` file.

Security Trimming

If you have been following the examples so far in this chapter, you might notice that one of the attributes available to a `<siteMapNode>` tag hasn't yet been discussed. The `roles` attribute is a powerful one that allows you to provide an authorization model to the items contained in the navigation system. This really means that you have the capability to display *only* the navigational items that a user is entitled to see and nothing more. The term commonly used for this behavior is *security trimming*. This section looks at how to apply security trimming to the application you are building in ASP.NET 3.5.

This capability is a good example of two ASP.NET 3.5 systems interacting with one another in the site navigation system. Security trimming works only when you have enabled the ASP.NET 3.5 role management system. This system is covered in more detail in Chapter 16. Be sure to check out this chapter because this section does not go into much detail about this system.

As an example of security trimming in your ASP.NET applications, this section shows you how to limit access to the navigation of your application's administration system only to users who are contained within a specific application role.

Setting Up Role Management for Administrators

The first step is to set up your application to handle roles. This is actually a pretty simple process. One easy way to accomplish this task is to open the ASP.NET Web Site Administration Tool for your application and enable role management directly in this Web-based tool. You can get to this administration tool by clicking the ASP.NET Configuration button in the menu of the Solution Explorer in Visual Studio. This button has the logo of a hammer and a globe.

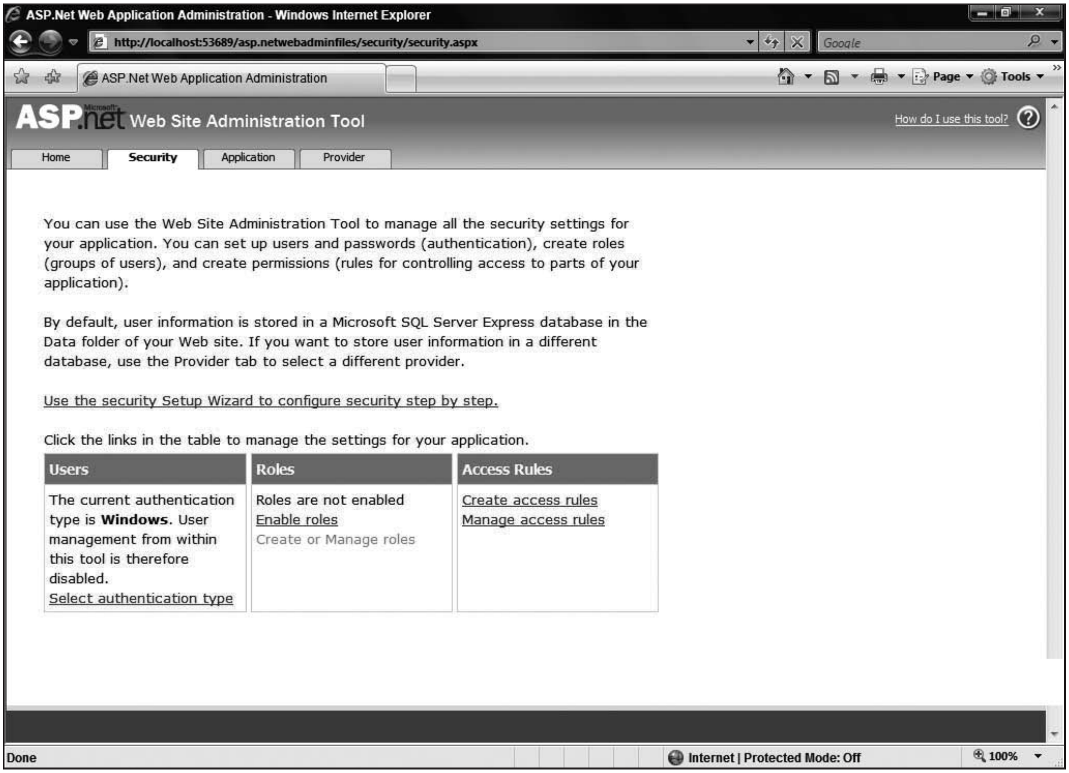
After the ASP.NET Web Site Administration Tool is launched, select the Security tab; this brings you to a screen where you can administer the membership and role management systems for your application.

First, you enable and build up the role management system, and then you also enable the membership system. The membership system is covered in detail in Chapter 16. After you turn on the membership system, you build some actual users in your application. You want a user to log in to your application and be assigned a specific role. This role assignment changes the site navigation system display.

The Security tab in the ASP.NET Web Site Administration Tool is presented in Figure 14-35.

On this page, you can easily enable the role management system by selecting the Enable roles link. After you have done this, you are informed that there are no roles in the system. To create the role that you need for the site navigation system, select the Create or Manage roles link. You are then presented with a page where you can create the administrator role. For this example, I named the role Admin.

After adding the Admin role, click the Back button and then select the authentication type that is utilized for the application. You want to make sure that you have selected the From the internet option. This enables you then to create a user in the system. By default, these users are stored in the Microsoft SQL Server Express Edition file that ASP.NET creates in your application. After you have selected the authentication type, you can then create the new user and place the user in the Admin role by making sure the role is selected (using a check box) on the screen where you are creating the user.


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:53689/asp.netwebadminfiles/security/security.aspx
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Done
Internet | Protected Mode: Off
100%

You can use the Web Site Administration Tool to manage all the security settings for your application. You can set up users and passwords (authentication), create roles (groups of users), and create permissions (rules for controlling access to parts of your application).

By default, user information is stored in a Microsoft SQL Server Express database in the Data folder of your Web site. If you want to store user information in a different database, use the Provider tab to select a different provider.

Use the security Setup Wizard to configure security step by step.

Click the links in the table to manage the settings for your application.

Users	Roles	Access Rules
The current authentication type is **Windows**. User management from within this tool is therefore disabled. Select authentication type	Roles are not enabled Enable roles Create or Manage roles	Create access rules Manage access rules

Figure 14-35

After you are satisfied that a user has been created and placed in the Admin role, you can check if the settings are appropriately set in the `web.config` file. This is presented in Listing 14-33.

Listing 14-33: The role management system enabled in the web.config file

```
<configuration>
   <system.web>

      <authentication mode="Forms" />
      <roleManager enabled="true" />

   </system.web>
</configuration>
```

Setting Up the Administrators' Section

The next step is to set up a page for administrators only. For this example, I named the page `AdminOnly.aspx`, and it contains only a simple string value welcoming administrators to the page. This page is locked down only for users who are contained in the Admin role. This is done by making the appropriate settings in the `web.config` file. This lockdown is shown in Listing 14-34.

Listing 14-34: Locking down the AdminOnly.aspx page in the web.config

```
<configuration>
  <location path="AdminOnly.aspx">
    <system.web>
      <authorization>
        <allow roles="Admin" />
        <deny users="*" />
      </authorization>
    </system.web>
  </location>
</configuration>
```

Now, because the `AdminOnly.aspx` page is accessible only to the users who are in the Admin role, the next step is to allow users to login to the application. The application demo accomplishes this task by creating a `Default.aspx` page that contains a Login server control as well as a TreeView control bound to a SiteMapDataSource control. This simple ASP.NET page is presented in Listing 14-35.

Listing 14-35: The Default.aspx page

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Main Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Login ID="Login1" runat="server">
        </asp:Login>
        <br />
        <asp:TreeView ID="TreeView1" runat="server"
         DataSourceID="SiteMapDataSource1" ShowLines="True">
        </asp:TreeView>
        <br />
        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
    </div>
    </form>
</body>
</html>
```

With the `Default.aspx` page in place, another change is made to the `Web.sitemap` file that was originally presented in Listing 14-1. For this example, you add a `<siteMapNode>` element that works with the new `AdminOnly.aspx` page. This node is presented here:

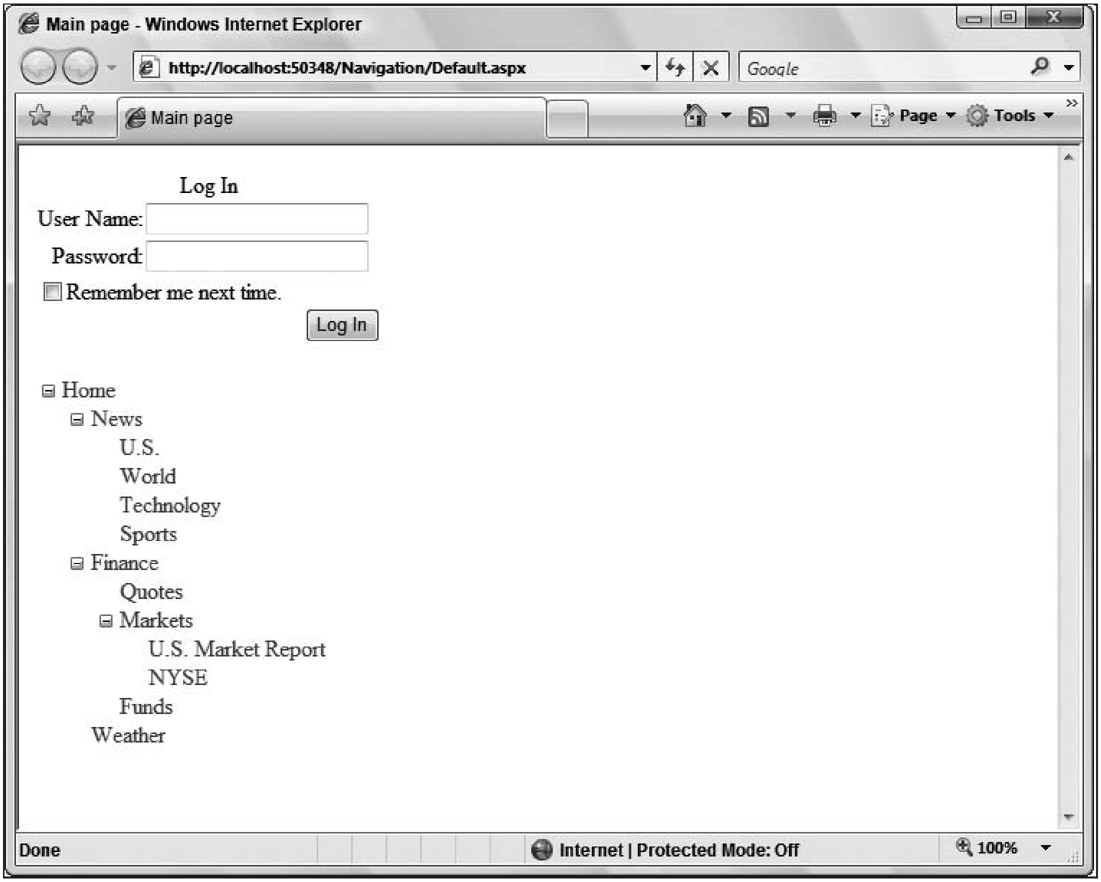
```
<siteMapNode title="Administration" description="The Administrators page"
 url="AdminOnly.aspx" roles="Admin" />
```

After all items are in place in your application, the next step is to enable security trimming for the site navigation system.

Enabling Security Trimming

By default, security trimming is disabled. Even if you start applying values to the `roles` attribute for any `<siteMapNode>` element in your `web.config` file, it does not work. To enable security trimming, you must fine-tune the provider declaration for the site navigation system.

To make the necessary changes to the XmlSiteMapProvider, you need to make these changes high up in the configuration chain, such as to the `machine.config` file or the default `web.config` file, or you can make the change lower down, such as in your application's `web.config` file. This example makes the change in the `web.config` file.


Main page - Windows Internet Explorer
http://localhost:50348/Navigation/Default.aspx
Google
Main page
Page
Tools
Log In
User Name:
Password:
☐ Remember me next time.
Log In
Done
Internet | Protected Mode: Off
100%

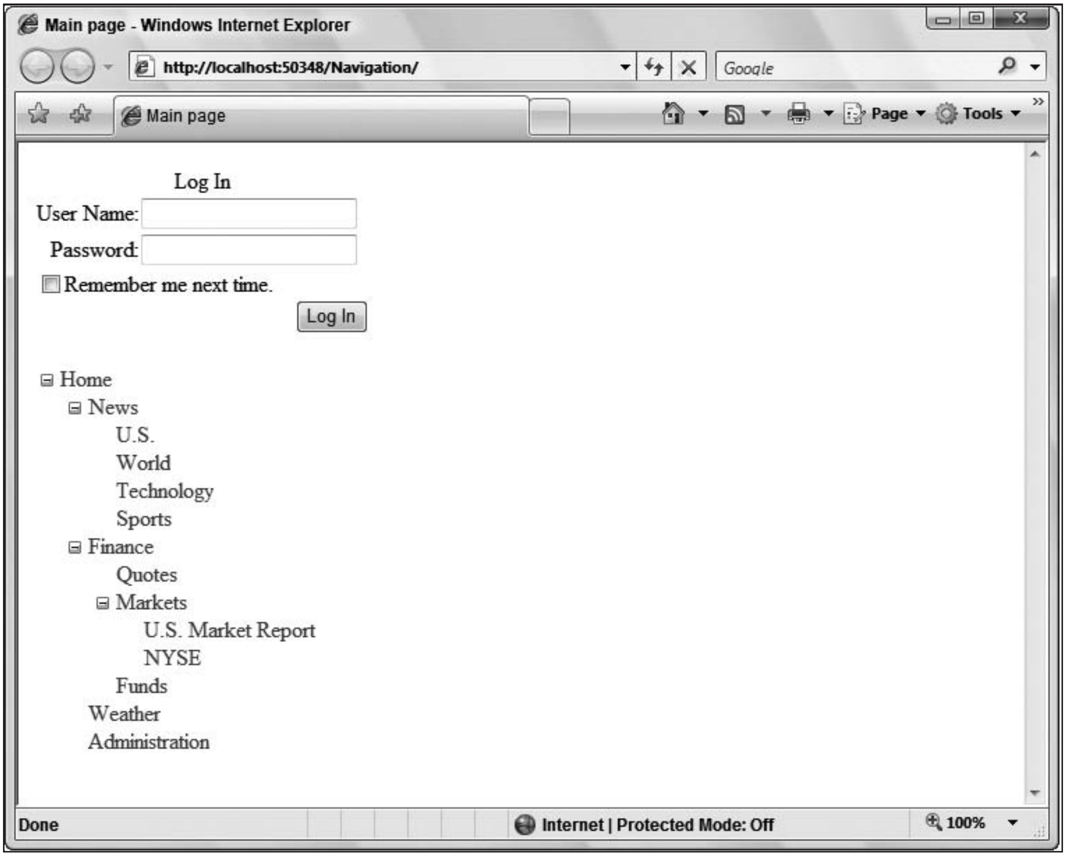
- ⊟ Home
  - ⊟ News
    - U.S.
    - World
    - Technology
    - Sports
  - ⊟ Finance
    - Quotes
    - ⊟ Markets
      - U.S. Market Report
      - NYSE
    - Funds
  - Weather

Figure 14-36

To alter the XmlSiteMapProvider in the `web.config` file, you first clear out the already declared instance. After it is cleared, you then redeclare a new instance of the XmlSiteMapProvider, but this time you enable security trimming, as illustrated in Listing 14-36.

Listing 14-36: Enabling security trimming in the provider

```
<configuration>
   <system.web>

      <siteMap>
        <providers>
          <clear />
          <add siteMapFile="web.sitemap" name="AspNetXmlSiteMapProvider"
              type="System.Web.XmlSiteMapProvider, System.Web, Version=2.0.0.0,
                 Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
               securityTrimmingEnabled="true" />
         </providers>
       </siteMap>

    </system.web>
 </configuration>
```


Main page - Windows Internet Explorer
http://localhost:50348/Navigation/
Google
Main page
Page
Tools
Done
Internet | Protected Mode: Off
100%

Log In
User Name:
Password:
☐ Remember me next time.
Log In

- Home
  - News
    - U.S.
    - World
    - Technology
    - Sports
  - Finance
    - Quotes
    - Markets
      - U.S. Market Report
      - NYSE
    - Funds
  - Weather
  - Administration

Figure 14-37

From this example, you can see that a new XmlSiteMapProvider is defined and the `securityTimmingEnabled` attribute is then set to `true` (shown in bold). With security trimming enabled, the roles attribute in the `<siteMapNode>` element is utilized in the site navigation system.

To test it out for yourself, run the `Default.aspx` page. You are first presented with a page that does not include the link to the administration portion of the page, as illustrated in Figure 14-36.

From this figure, you can see that the Administration link is not present in the TreeView control. Now, however, log in to the application as a user contained in the Admin role. You then see that, indeed, the site navigation has changed to reflect the role of the user, as presented in Figure 14-37.

Security trimming is an ideal way of automatically altering the site navigation that is presented to users based upon the roles they have in the role management system.

Nesting SiteMap Files

You are not required to place all your site navigation within a single `Web.sitemap` file. In fact, you can spread it out into multiple `.sitemap` files if you wish and then bring them all together into a single `.sitemap` file — also known as *nesting* `.sitemap` files.

For instance, suppose you are using the sitemap file from Listing 14-1 and you have a pretty large amount of site navigation to add under the area of *Entertainment*. You could put all this new information in the current `Web.sitemap` file, or you could keep all the Entertainment links in another sitemap file and just reference that in the main sitemap file.

In the simplest case, nesting sitemap files is easily achievable. To see it in action, create a new `.sitemap` file (called `Entertainment.sitemap`) and place this file in your application. An example `Entertainment.sitemap` file is presented in Listing 14-37.

Listing 14-37: The Entertainment.sitemap

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="Entertainment.aspx" title="Entertainment"
     description="The Entertainment Page">
        <siteMapNode url="Movies.aspx" title="Movies"
         description="The Latest in Movies" />
        <siteMapNode url="Fashion.aspx" title="Fashion"
         description="The Latest in Fashion" />
    </siteMapNode>
</siteMap>
```

You can place the `Entertainment.sitemap` in the root directory where you also have the main `Web.sitemap` file. Now, working from the sitemap file that from the earlier Listing 14-1, you make the following addition to the bottom of the list as presented in Listing 14-38.

Listing 14-38: Additions to the Web.sitemap file

```
<siteMapNode siteMapFile="Entertainment.sitemap" />
```

Instead of using the standard `url`, `title`, and `description` attributes, you just point to the other sitemap file to be included in this main sitemap file using the `siteMapFile` attribute. Running this page gives you results similar to those presented in Figure 14-38.

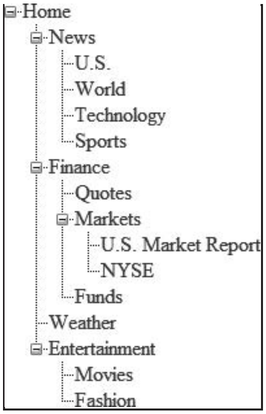


Figure 14-38

Another approach to nesting sitemap files is to build a second provider in the sitemap provider model definitions and to then use the `provider` attribute within the `<siteMapNode>` element to reference this declaration. To accomplish this task, you first add a new sitemap provider reference in the `web.config` file. This is illustrated in Listing 14-39.

Listing 14-39: Using another provider in the same Web.sitemap file

```
<configuration>
   <system.web>

      <siteMap>
        <providers>
          <add siteMapFile="Entertainment.sitemap" name="AspNetXmlSiteMapProvider2"
              type="System.Web.XmlSiteMapProvider, System.Web, Version=2.0.0.0,
                 Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
        </providers>
      </siteMap>

   </system.web>
</configuration>
```

From this bit of code, you can see that a second provider is defined. Defining a second sitemap provider does not mean that you have to use the `<clear />` element in the `<provider>` section, but instead, you simply define a new provider that has a new name. In this case, the name of the provider is `AspNetXmlSiteMapProvider2`. Also, within this provider definition, the `siteMapFile` attribute is used to point to the name of the sitemap file that should be utilized.

With this in place, you can then reference this declaration by using the `provider` attribute within the `<siteMapNode>` element of the `Web.sitemap` file. To add the `Entertainment.sitemap` file in this manner, your `<siteMapNode>` element should take the form presented in Listing 14-40.

Listing 14-40: Using a second provider in the Web.sitemap file

```
<siteMapNode provider="AspNetXmlSiteMapProvider2" />
```

This gives you the same results as those shown in Figure 14-38. Besides providing another way of nesting sitemap files, you gain a lot of power using the `provider` attribute. If you build a new sitemap provider that pulls sitemap navigation information from another source (rather than from an XML file), you can mix those results in the main `Web.sitemap` file. The end result could have items that come from two or more completely different data sources.

Summary

This chapter introduced the navigation mechanics that ASP.NET 3.5 provides. At the core of the navigation capabilities is the power to detail the navigation structure in an XML file, which can then be utilized by various navigation controls — such as the new TreeView and SiteMapPath controls.

The powerful functionality that the navigation capabilities provide saves you a tremendous amount of coding time.

In addition to showing you the core infrastructure for navigation in ASP.NET 3.5, this chapter also described both the TreeView and SiteMapPath controls and how to use them throughout your applications. The great thing about these controls is that, right out of the box, they can richly display your navigation hierarchy and enable the end user to work through the site easily. In addition, these controls are easily changeable so that you can go beyond the standard appearance and functionality that they provide.

Along with the TreeView server control, this chapter also looked at the Menu server control. You will find a lot of similarities between these two controls as they both provide a means to look at hierarchical data.

Finally, this chapter looked at how to achieve URL mapping, as well as how to localize your `Web.sitemap` files and filter the results of the site navigation contents based upon a user's role in the role management system.

15

Personalization

Many Web applications must be customized with information that is specific to the end user who is presently viewing the page. In the past, the developer usually provided storage of personalization properties for end users viewing the page by means of cookies, the `Session` object, or the `Application` object. Cookies enabled storage of persistent items so that when the end user returned to a Web page, any settings related to him were retrieved in order to be utilized again by the application. Cookies are not the best way to approach persistent user data storage, however, mainly because they are not accepted by all computers and also because a crafty end user can easily alter them.

As you will see in Chapter 16, ASP.NET membership and role management capabilities are ways that ASP.NET can conveniently store information about the user. How can you, as the developer, use the same mechanics to store custom information?

ASP.NET 3.5 provides you with an outstanding feature — *personalization*. The ASP.NET personalization engine provided with this latest release can make an automatic association between the end user viewing the page and any data points stored for that user. The personalization properties that are maintained on a per-user basis are stored on the server and not on the client. These items are conveniently placed in a data store of your choice (such as Microsoft's SQL Server) and, therefore, the end user can then access these personalization properties on later site visits.

This feature is an ideal way to start creating highly customizable and user-specific sites without building any of the plumbing beforehand. In this case, the plumbing has been built for you! The personalization feature is yet another way that the ASP.NET team is making developers more productive and their jobs easier.

The Personalization Model

The personalization model provided with ASP.NET 3.5 is simple and, as with most items that come with ASP.NET, it is an extensible model as well. Figure 15-1 shows a simple diagram that outlines the personalization model.

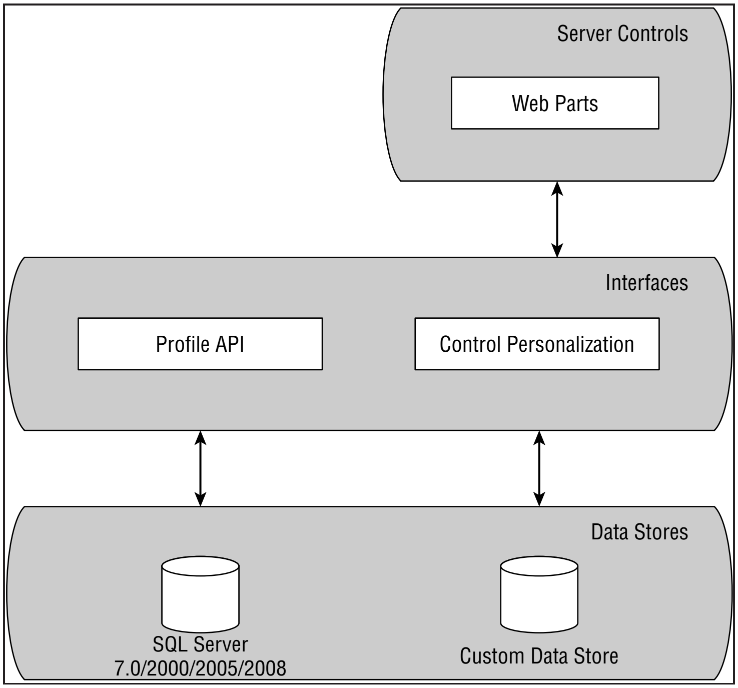


Figure 15-1

From this diagram, you can see the three layers in this model. First, look at the middle layer of the personalization model — the Personalization Services layer. This layer contains the Profile API. This new Profile API layer enables you to program your end user's data points into one of the lower-layer data stores. Also included in this layer are the server control personalization capabilities, which are important for the Portal Framework and the use of Web Parts. The Portal Framework and Web Parts are discussed in Chapter 17.

Although certain controls built into ASP.NET can utilize the personalization capabilities for storing information about the page settings, you can also use this new engine to store your own data points. As with Web Parts, these points can be used within your ASP.NET pages.

Below the Personalization Services layer, you find the default personalization data provider for working with Microsoft's SQL Server 2008, 2005, or 2000, as well as the new Microsoft SQL Server Express Edition files. You are not limited to just this one data store when applying the personalization features of ASP.NET 3.5; you can also extend the model and create a custom data provider for the personalization engine.

> *You can read about how to create your own providers in Chapter 13.*

Now that you have looked briefly at the personalization model, you can begin using it by creating some stored personalization properties that can be used later within your applications.

Creating Personalization Properties

The nice thing about creating custom personalization properties is that you can do it so easily. After these properties are created, you gain the capability to have strongly typed access to them. It is also possible to create personalization properties that are used only by authenticated users, and also some that anonymous users can utilize. These data points are powerful — mainly because you can start using them immediately in your application without building any underlying infrastructures to support them. The first step is to create some simple personalization properties. Later, you learn how to use these personalization properties within your application.

Adding a Simple Personalization Property

The first step is to decide what data items from the user you are going to store. For this example, create a few items about the user that you can use within your application; assume that you want to store the following information about the user:

- ❑ First name
- ❑ Last name
- ❑ Last visited
- ❑ Age
- ❑ Membership Status

ASP.NET has a heavy dependency on storing configurations inside XML files, and the ASP.NET 3.5 personalization engine is no different. All these customization points concerning the end user are defined and stored within the `web.config` file of the application. This is illustrated in Listing 15-1.

Listing 15-1: Creating personalization properties in the web.config file

```
<configuration>
  <system.web>

     <profile>

        <properties>

           <add name="FirstName" />
           <add name="LastName" />
           <add name="LastVisited" />
           <add name="Age" />
           <add name="Member" />

        </properties>

      </profile>

      <authentication mode="Windows" />

  </system.web>
</configuration>
```

Within the `web.config` file and nested within the `<system.web>` section of the file, you create a `<profile>` section in order to work with the ASP.NET 3.5 personalization engine. Within this `<profile>` section of the `web.config` file, you create a `<properties>` section. In this section, you can define all the properties you want the personalization engine to store.

From this code example, you can see that it is rather easy to define simple properties using the `<add>` element. This element simply takes the `name` attribute, which takes the name of the property you want to persist.

You start out with the assumption that accessing the page you will build with these properties is already authenticated using Windows authentication (you can read more on authentication and authorization in the next chapter). Later in this chapter, you look at how to apply personalization properties to anonymous users as well. The capability to apply personalization properties to anonymous users is disabled by default (for good reasons).

After you have defined these personalization properties, it is just as easy to use them as it is to define them. The next section looks at how to use these definitions in an application.

Using Personalization Properties

Now that you have defined the personalization properties in the `web.config` file, it is possible to use these items in code. For example, you can create a simple form that asks for some of this information from the end user. On the `Button_Click` event, the data is stored in the personalization engine. Listing 15-2 shows an example of this.

Listing 15-2: Using the defined personalization properties

VB

```
<%@ Page Language="VB" %>

<script runat="server">
  Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    If Page.User.Identity.IsAuthenticated Then
       Profile.FirstName = TextBox1.Text
       Profile.LastName = TextBox2.Text
       Profile.Age = TextBox3.Text
       Profile.Member = Radiobuttonlist1.SelectedItem.Text
       Profile.LastVisited = DateTime.Now().ToString()

       Label1.Text = "Stored information includes:<p>" & _
          "First name: " & Profile.FirstName & _
          "<br>Last name: " & Profile.LastName & _
          "<br>Age: " & Profile.Age & _
          "<br>Member: " & Profile.Member & _
          "<br>Last visited: " & Profile.LastVisited
    Else
       Label1.Text = "You must be authenticated!"
    End If
  End Sub
</script>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Storing Personalization</title>
</head>
<body>
   <form id="form1" runat="server">
       <p>First Name:
       <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox></p>
       <p>Last Name:
       <asp:TextBox ID="TextBox2" Runat="server"></asp:TextBox></p>
       <p>Age:
       <asp:TextBox ID="TextBox3" Runat="server" Width="50px"
        MaxLength="3"></asp:TextBox></p>
       <p>Are you a member?
       <asp:RadioButtonList ID="Radiobuttonlist1" Runat="server">
           <asp:ListItem Value="1">Yes</asp:ListItem>
           <asp:ListItem Value="0" Selected="True">No</asp:ListItem>
       </asp:RadioButtonList></p>
       <p><asp:Button ID="Button1" Runat="server" Text="Submit"
           OnClick="Button1_Click" />
       </p>
       <hr /><p>
       <asp:Label ID="Label1" Runat="server"></asp:Label></p>
   </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
      if (Page.User.Identity.IsAuthenticated)
      {
        Profile.FirstName = TextBox1.Text;
        Profile.LastName = TextBox2.Text;
        Profile.Age = TextBox3.Text;
        Profile.Member = Radiobuttonlist1.SelectedItem.Text;
        Profile.LastVisited = DateTime.Now.ToString();

        Label1.Text = "Stored information includes:<p>" +
           "First name: " + Profile.FirstName +
           "<br>Last name: " + Profile.LastName +
           "<br>Age: " + Profile.Age +
           "<br>Member: " + Profile.Member +
           "<br>Last visited: " + Profile.LastVisited;
      }
      else
      {
        Label1.Text = "You must be authenticated!";
      }
    }
</script>
```

This is similar to the way you worked with the `Session` object in the past, but note that the personalization properties you are storing and retrieving are not key based. Therefore, when working with them you do not need to remember key names.

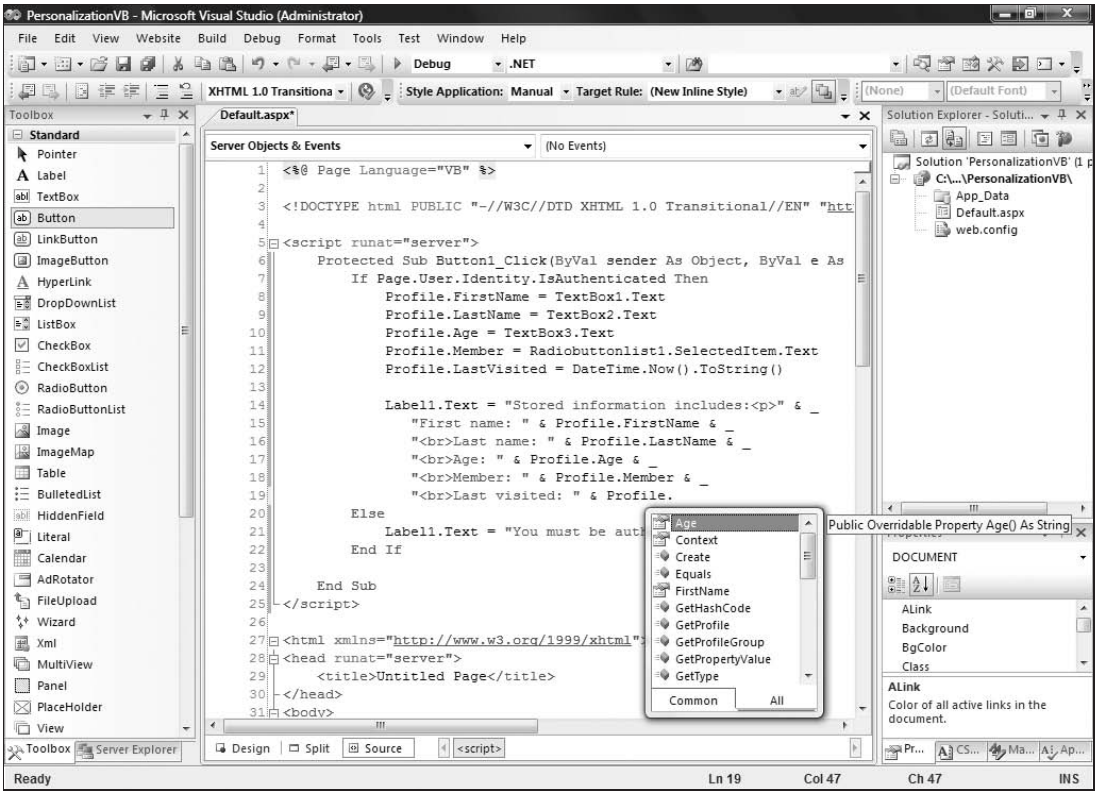
All items stored by the personalization system are type cast to a particular .NET data type. By default, these items are stored as type `String`, and you have early-bound access to the items stored. To store an item, you simply populate the personalization property directly using the `Profile` object:

```
Profile.FirstName = TextBox1.Text
```

To retrieve the same information, you simply grab the appropriate property of the `Profile` class as shown here:

```
Label1.Text = Profile.FirstName
```

The great thing about using the `Profile` class and all the personalization properties defined in the code view is that this method provides IntelliSense as you build your pages. When you are working with the `Profile` class in this view, all the items you define are listed as available options through the IntelliSense feature, as illustrated in Figure 15-2.

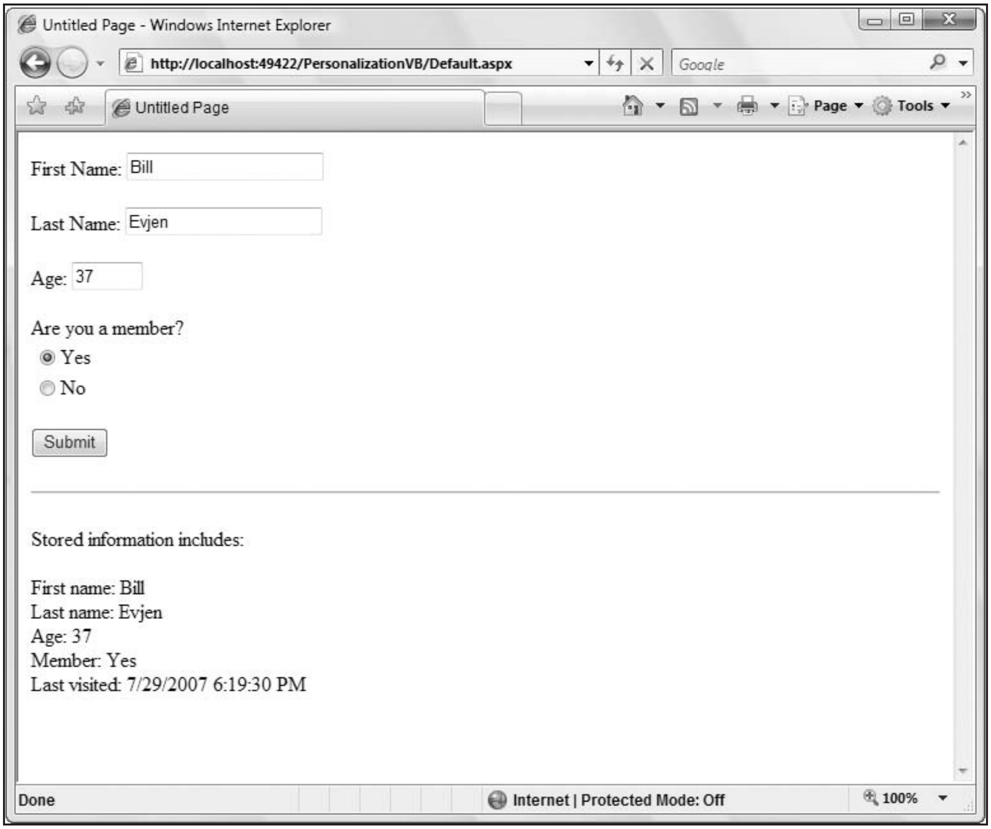

PersonalizationVB - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Format Tools Test Window Help
Debug
.NET
XHTML 1.0 Transitiona
Style Application: Manual
Target Rule: (New Inline Style)
(None)
(Default Font)
Toolbox
Standard
Pointer
Label
TextBox
Button
LinkButton
ImageButton
HyperLink
DropDownList
ListBox
CheckBox
CheckBoxList
RadioButton
RadioButtonList
Image
ImageMap
Table
BulletedList
HiddenField
Literal
Calendar
AdRotator
FileUpload
Wizard
Xml
MultiView
Panel
PlaceHolder
View
Toolbox
Server Explorer
Default.aspx*
Server Objects & Events
(No Events)
Age
Context
Create
Equals
FirstName
GetHashCode
GetProfile
GetProfileGroup
GetPropertyValue
GetType
Common
All
Public Overridable Property Age() As String
Solution Explorer - Soluti...
Solution 'PersonalizationVB' (1 p
C:\...\PersonalizationVB\
App_Data
Default.aspx
web.config
DOCUMENT
ALink
Background
BgColor
Class
ALink
Color of all active links in the document.
Design
Split
Source
<script>
Ready
Ln 19
Col 47
Ch 47
INS

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "htt

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
        If Page.User.Identity.IsAuthenticated Then
            Profile.FirstName = TextBox1.Text
            Profile.LastName = TextBox2.Text
            Profile.Age = TextBox3.Text
            Profile.Member = Radiobuttonlist1.SelectedItem.Text
            Profile.LastVisited = DateTime.Now().ToString()

            Label1.Text = "Stored information includes:<p>" & _
               "First name: " & Profile.FirstName & _
               "<br>Last name: " & Profile.LastName & _
               "<br>Age: " & Profile.Age & _
               "<br>Member: " & Profile.Member & _
               "<br>Last visited: " & Profile.
        Else
            Label1.Text = "You must be aut
        End If

    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml"
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
```

Figure 15-2

All these properties are accessible in IntelliSense because the `Profile` class is hidden and dynamically compiled behind the scenes whenever you save the personalization changes made to the `web.config` file. After these items are saved in the `web.config` file, these properties are available to you throughout your application in a strongly typed manner.

When run, the page from Listing 15-2 produces the results shown in Figure 15-3.


Untitled Page - Windows Internet Explorer
http://localhost:49422/PersonalizationVB/Default.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

First Name: Bill

Last Name: Evjen

Age: 37

Are you a member?

● Yes

○ No

Submit

Stored information includes:

First name: Bill
Last name: Evjen
Age: 37
Member: Yes
Last visited: 7/29/2007 6:19:30 PM

Figure 15-3

In addition to using early-bound access techniques, you can also use late-bound access for the items that you store in the personalization engine. This technique is illustrated in Listing 15-3.

Listing 15-3: Using late-bound access

VB

```
Dim myFirstName As String

myFirstName = Profile.GetPropertyValue("FirstName").ToString()
```

Continued

C#

```
string myFirstName;

myFirstName = Profile.GetPropertyValue("FirstName").ToString();
```

Whether it is early-bound access or late-bound access, you can easily store and retrieve personalization properties for a particular user using this capability afforded by ASP.NET 3.5. All this is done in the personalization engine's simplest form — now take a look at how you can customize for specific needs in your applications.

Adding a Group of Personalization Properties

If you want to store a large number of personalization properties about a particular user, remember that you are not just storing personalization properties for a particular page, but for the *entire* application. This means that items you have stored about a particular end user somewhere in the beginning of the application can be retrieved later for use on any other page within the application. Because different sections of your Web applications store different personalization properties, you sometimes end up with a large collection of items to be stored and then made accessible.

To make it easier not only to store the items, but also to retrieve them, the personalization engine enables you to store your personalization properties in groups. This is illustrated in Listing 15-4.

Listing 15-4: Creating personalization groups in the web.config file

```
<configuration>
  <system.web>

     <profile>

        <properties>

           <add name="FirstName" />
           <add name="LastName" />
           <add name="LastVisited" />
           <add name="Age" />

           <group name="MemberDetails">
              <add name="Member" />
              <add name="DateJoined" />
              <add name="PaidDuesStatus" />
              <add name="Location" />
           </group>

           <group name="FamilyDetails">
              <add name="MarriedStatus" />
              <add name="DateMarried" />
              <add name="NumberChildren" />
              <add name="Location" />
           </group>

         </properties>
```

```
    </profile>

    <authentication mode="Windows" />

  </system.web>
</configuration>
```

From the code in Listing 15-4, which is placed within the `web.config` file, you can see that two groups are listed. The first group is the `MemberDetails` group, which has four specific items defined; the second group — `FamilyDetails` — has three other related items defined. Personalization groups are defined using the `<group>` element within the `<properties>` definition. The name of the group is specified using the `name` attribute, just as you specify the `<add>` element. You can have as many groups defined as you deem necessary or as have been recommended as good practice to employ.

Using Grouped Personalization Properties

From Listing 15-4, you can also see that some items are not defined in any particular group. It is possible to mix properties defined from within a group with those that are not. The items not defined in a group in Listing 15-4 can still be accessed in the manner illustrated previously:

```
Label1.Text = Profile.FirstName
```

Now, concerning working with personalization groups, you can access your defined items in a logical manner using nested namespaces:

```
Label1.Text = Profile.MemberDetails.DateJoined

Label2.Text = Profile.FamilyDetails.MarriedStatus
```

From this example, you can see that two separate items from each of the defined personalization groups were accessed in a logical manner. When you study the defined properties in the `web.config` file of your application, you can see that each of the groups in the example has a property with the same name — `Location`. This is possible because they are defined using personalization groups. With this structure, it is now possible to get at each of the `Location` properties by specifying the appropriate group:

```
Label1.Text = Profile.MemberDetails.Location

Label2.Text = Profile.FamilyDetails.Location
```

Defining Types for Personalization Properties

By default, when you store personalization properties, these properties are created as type `System.String`. It is quite easy, however, to change the type to another type altogether through configuration settings within the `web.config` file. To define the name of the personalization property along with its appropriate type, you use the `type` attribute of the `<add>` element contained within the `<properties>` section, as shown in Listing 15-5.

Listing 15-5: Defining types for personalization properties

```
<properties>

   <add name="FirstName" type="System.String" />
```

Continued

```
        <add name="LastName" type="System.String" />
        <add name="LastVisited" type="System.DateTime" />
        <add name="Age" type="System.Int32" />
        <add name="Member" type="System.Boolean" />

    </properties>
```

The first two properties, `FirstName` and `LastName`, are cast as type `System.String`. This is not actually required. Even if you omitted this step, they would still be cast as type `String` because that is the default type of any property defined in the personalization system (if no other type is defined). The next personalization property is the `LastVisited` property, which is defined as type `System.DateTime` and used to store the date and time of the end user's last visit to the page. Beyond that, you can see the rest of the personalization properties are defined using a specific .NET data type.

This is the preferred approach because it gives you type-checking capabilities as you code your application and use the personalization properties you have defined.

Using Custom Types

As you can see from the earlier examples that show you how to define types for the personalization properties, it is quite simple to define properties and type cast them to particular data types that are available in the .NET Framework. Items such as `System.Integer`, `System.String`, `System.DateTime`, `System.Byte`, and `System.Boolean` are easily defined within the `web.config` file. But how do you go about defining complex types?

Personalization properties that utilize custom types are just as easy to define as personalization properties that use simple types. Custom types give you the capability to store complex items such as shopping cart information or other status information from one use of the application to the next. Listing 15-6 first shows a class, `ShoppingCart`, which you use later in one of the personalization property definitions.

Listing 15-6: Creating a class to use as a personalization type

VB

```
<Serializable()> _
Public Class ShoppingCart
    Private PID As String
    Private CompanyProductName As String
    Private Number As Integer
    Private Price As Decimal
    Private DateAdded As DateTime

    Public Property ProductID() As String
        Get
            Return PID
        End Get
        Set(ByVal value As String)
            PID = value
        End Set
    End Property

    Public Property ProductName() As String
        Get
```

Continued

```
            Return CompanyProductName
        End Get
        Set(ByVal value As String)
            CompanyProductName = value
        End Set
    End Property

    Public Property NumberSelected() As Integer
        Get
            Return Number
        End Get
        Set(ByVal value As Integer)
            Number = value
        End Set
    End Property

    Public Property ItemPrice() As Decimal
        Get
            Return Price
        End Get
        Set(ByVal value As Decimal)
            Price = value
        End Set
    End Property

    Public Property DateItemAdded() As DateTime
        Get
            Return DateAdded
        End Get
        Set(ByVal value As DateTime)
            DateAdded = value
        End Set
    End Property
End Class
```

C#

```
using System;

[Serializable]
public class ShoppingCart
{
    private string PID;
    private string CompanyProductName;
    private int Number;
    private decimal Price;
    private DateTime DateAdded;

    public ShoppingCart() {}

    public string ProductID
    {
        get {return PID;}
        set {PID = value;}
    }
```

Continued

```
        public string ProductName
        {
            get { return CompanyProductName; }
            set { CompanyProductName = value; }
        }

        public int NumberSelected
        {
            get { return Number; }
            set { Number = value; }
        }

        public decimal ItemPrice
        {
            get { return Price; }
            set { Price = value; }
        }

        public DateTime DateItemAdded
        {
            get { return DateAdded; }
            set { DateAdded = value; }
        }
    }
```

This simple shopping cart construction can now store the end user's shopping cart basket as the user moves around on an e-commerce site. The basket can even be persisted when the end user returns to the site at another time. Be sure to note that the class requires a `Serializable` attribute preceding the class declaration to ensure proper transformation to XML or binary.

Look at how you would specify from within the `web.config` file that a personalization property is this complex type, such as a `ShoppingCart` type. This is illustrated in Listing 15-7.

Listing 15-7: Using complex types for personalization properties

```
<properties>

   <add name="FirstName" type="System.String" />
   <add name="LastName" type="System.String" />
   <add name="LastVisited" type="System.DateTime" />
   <add name="Age" type="System.Int32" />
   <add name="Member" type="System.Boolean" />
   <add name="Cart" type="ShoppingCart" serializeAs="Binary" />

</properties>
```

Just as the basic data types are stored in the personalization data stores, this construction allows you to easily store custom types and to have them serialized into the end data store in the format you choose. In this case, the `ShoppingCart` object is serialized as a binary object into the data store. The `SerializeAs` attribute can take the values defined in the following list:

- **Binary:** Serializes and stores the object as binary data within the chosen data store.
- **ProviderSpecific:** Stores the object based upon the direction of the provider. This simply means that instead of the personalization engine determining the serialization of the object, the serialization is simply left up to the personalization provider specified.

- ❑ **String:** The default setting. Stores the personalization properties as a string inside the chosen data store.
- ❑ **XML:** Takes the object and serializes it into an XML format before storing it in the chosen data store.

Providing Default Values

In addition to defining the data types of the personalization properties, you can also define their default values directly in the `web.config` file. By default, the personalization properties you create do not have a value, but you can easily change this using the `defaultValue` attribute of the `<add>` element. Defining default values is illustrated in Listing 15-8.

Listing 15-8: Defining default values for personalization properties

```
<properties>

   <add name="FirstName" type="System.String" />
   <add name="LastName" type="System.String" />
   <add name="LastVisited" type="System.DateTime" />
   <add name="Age" type="System.Integer" />
   <add name="Member" type="System.Boolean" defaultValue="false" />

</properties>
```

From this example, you can see that only one of the personalization properties is provided with a default value. The last personalization property, `Member` in this example, is given a default value of `false`. This means that when you add a new end user to the personalization property database, `Member` is defined instead of remaining a blank value within the system.

Making Personalization Properties Read-Only

It is also possible to make personalization properties read-only. To do this, you simply add the `readOnly` attribute to the `<add>` element:

```
<add name="StartDate" type="System.DateTime" readOnly="true" />
```

To make the personalization property a read-only property, you give the `readOnly` attribute a value of `true`. By default, this property is set to `false`.

Anonymous Personalization

A great feature in ASP.NET enables anonymous end users to utilize the personalization features it provides. This is important if a site requires registration of some kind. In these cases, end users do not always register for access to the greater application until they have first taken advantage of some of the basic services. For instance, many e-commerce sites allow anonymous end users to shop a site and use the site's shopping cart before the shoppers register with the site.

Enabling Anonymous Identification of the End User

By default, anonymous personalization is turned off because it consumes database resources on popular sites. Therefore, one of the first steps in allowing anonymous personalization is to turn on this feature using the appropriate setting in the `web.config` file. You also need to make some changes regarding how the properties are actually defined in the `web.config` file and to determine if you are going to allow anonymous personalization for your application.

As shown in Listing 15-9, you can turn on anonymous identification to enable the personalization engine to identify the unknown end users using the `<anonymousIdentification>` element.

Listing 15-9: Allowing anonymous identification

```
<configuration>
  <system.web>

       <anonymousIdentification enabled="true" />

  </system.web>
</configuration>
```

To enable anonymous identification of the end users who might visit your applications, you add an `<anonymousIdentification>` element to the `web.config` file within the `<system.web>` nodes. Then within the `<anonymousIdentification>` element, you use the `enabled` attribute and set its value to `true`. Remember that by default, this value is set to `false`.

When anonymous identification is turned on, ASP.NET uses a unique identifier for each anonymous user who comes to the application. This identifier is sent with each and every request, although after the end user becomes authenticated by ASP.NET, the identifier is removed from the process.

For an anonymous user, information is stored by default as a cookie on the end user's machine. Additional information (the personalization properties that you enable for anonymous users) is stored in the specified data store on the server.

To see this in action, turn off the Windows Authentication for your example application and, instead, use Forms Authentication. This change is demonstrated in Listing 15-10.

Listing 15-10: Turning off Windows Authentication and using Forms Authentication

```
<configuration>
  <system.web>

       <anonymousIdentification enabled="true" />

       <authentication mode="Forms" />

  </system.web>
</configuration>
```

With this in place, if you run the page from the earlier example in Listing 15-2, you see the header presented in Listing 15-11.

Listing 15-11: Setting an anonymous cookie in the HTTP header

```
HTTP/1.1 200 OK
Server: ASP.NET Development Server/8.0.0.0
Date: Sat, 11 Feb 2008 19:23:37 GMT
X-AspNet-Version: 2.0.50727
Set-Cookie: .ASPXANONYMOUS=UH5CftJlxgEkAAAAZTJkN2I3YjUtZDhkOS00NDE2LWFlYjEtOTVj
    MjVmMzMxZWRmHoBU
As9A055rziDrMQ1Hu_fC_hM1; expires=Sat, 22-Apr-2008 06:03:36 GMT; path=/; HttpOnly
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1419
Connection: Close
```

From this HTTP header, you can see that a cookie — `.ASPXANONYMOUS` — is set to a hashed value for later retrieval by the ASP.NET personalization system.

Changing the Name of the Cookie for Anonymous Identification

Cookies are used by default under the cookie name `.ASPXANONYMOUS`. You can change the name of this cookie from the `<anonymousIdentification>` element in the `web.config` file by using the `cookieName` attribute, as shown in Listing 15-12.

Listing 15-12: Changing the name of the cookie

```
<configuration>
  <system.web>

       <anonymousIdentification
           enabled="true"
           cookieName=".ASPXEvjenWebApplication" />

  </system.web>
</configuration>
```

Changing the Length of Time the Cookie Is Stored

Also, by default, the cookie stored on the end user's machine is stored for 100,000 minutes (which is almost 70 days). If you want to change this value, you do it within this `<anonymousIdentification>` element using the `cookieTimeout` attribute, as shown in Listing 15-13.

Listing 15-13: Changing the length of time the cookie is stored

```
<configuration>
  <system.web>

       <anonymousIdentification
           enabled="true"
           cookieTimeout="1440" />

  </system.web>
</configuration>
```

In this case, the `cookieTimeout` value was changed to `1440` — meaning 1,440 minutes (or one day). This would be ideal for something like a shopping cart where you do not want to persist the identification of the end user too long.

Changing How the Identifiers Are Stored

Although anonymous identifiers are stored through the use of cookies, you can also easily change this. Cookies are, by far, the preferred way to achieve identification, but you can also do it without the use of cookies. Other options include using the URI or device profiles. Listing 15-14 shows an example of using the URI to place the identifiers.

Listing 15-14: Specifying how cookies are stored

```
<configuration>
  <system.web>

       <anonymousIdentification
           enabled="true"
           cookieless="UseUri" />

  </system.web>
</configuration>
```

Besides `UseUri`, other options include `UseCookies`, `AutoDetect`, and `UseDeviceProfile`. The following list reviews each of the options:

- ☐ `UseCookies`: This is the default setting. If you set no value, ASP.NET assumes this is the value. `UseCookies` means that a cookie is placed on the end user's machine for identification.
- ☐ `UseUri`: This value means that a cookie *will not* be stored on the end user's machine, but instead the unique identifier will be munged within the URL of the page. This is the same approach used for cookieless sessions in ASP.NET 1.0/1.1. Although this is great if developers want to avoid sticking a cookie on an end user's machine, it does create strange looking URLs and can be an issue when an end user bookmarks pages for later retrieval.
- ☐ `AutoDetect`: Using this value means that you are letting the ASP.NET engine decide whether to use cookies or use the URL approach for the anonymous identification. This is done on a per-user basis and performs a little worse than the other two options. ASP.NET must check the end user before deciding which approach to use. My suggestion is to use `AutoDetect` instead of `UseUri` if you absolutely must allow for end users who have cookies turned off (which is rare these days).
- ☐ `UseDeviceProfile`: Configures the identifier for the device or browser that is making the request.

Looking at the Anonymous Identifiers Stored

In order to make the anonymous identifiers unique, a globally unique GUID is used. You can also now grab hold of this unique identifier for your own use. In order to retrieve the GUID, the `Request` object has been enhanced with an `AnonymousID` property. The `AnonymousID` property returns a value of type

`String`, which can be used in your code as shown here:

```
Label1.Text = Request.AnonymousID
```

Working with Anonymous Identification

In working with the creation of anonymous users, be aware of an important event which you can use form your `Global.asax` file that can be used for managing the process:

- ❑ `AnonymousIdentification_Creating`

By using the `AnonymousIdentification_Creating` event, you can work with the identification of the end user as it occurs. For instance, if you do not want to use GUIDs for uniquely identifying the end user, you can change the identifying value from this event instead.

To do so, create the event using the event delegate of type `AnonymousIdentificationEventArgs`, as illustrated in Listing 15-15.

Listing 15-15: Changing the unique identifier of the anonymous user

VB

```
Public Sub AnonymousIdentification_Creating(ByVal sender As Object, _
   ByVal e As AnonymousIDentificationEventArgs)

        e.AnonymousID = "Bubbles " & DateTime.Now()

End Sub
```

C#

```
public void AnonymousIdentification_Creating(object sender,
   AnonymousIDentificationEventArgs e)
{
       e.AnonymousID = "Bubbles " + DateTime.Now;
}
```

The `AnonymousIdentificationEventArgs` event delegate exposes an `AnonymousID` property that assigns the value used to uniquely identify the anonymous user. Now, instead of a GUID to uniquely identify the anonymous user as

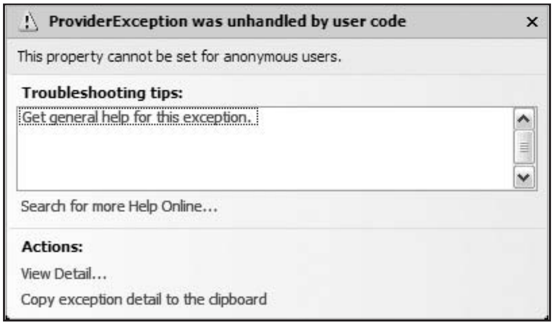
```
d13fafec-244a-4d21-9137-b213236ebedb
```

the `AnonymousID` property is changed within the `AnonymousIdentification_Creating` event to

```
Bubbles 2/10/2008 2:07:33 PM
```

Anonymous Options for Personalization Properties

If you have tried to get the anonymous capability working, you might have gotten the error shown in Figure 15-4.



ProviderException was unhandled by user code

This property cannot be set for anonymous users.

Troubleshooting tips:
Get general help for this exception.
Search for more Help Online...
Actions:
View Detail...
Copy exception detail to the clipboard

Figure 15-4

To get your application to work with anonymous users, you have to specify which personalization properties you wish to enable for the anonymous users visiting your pages. This is also done through the `web.config` file by adding the `allowAnonymous` attribute to the `<add>` element of the properties you have defined within the `<properties>` section (see Listing 15-16).

Listing 15-16: Turning on anonymous capabilities personalization properties

```
<properties>

   <add name="FirstName" type="System.String" />
   <add name="LastName" type="System.String" />
   <add name="LastVisited" type="System.DateTime" allowAnonymous="true" />
   <add name="Age" type="System.Integer" />
   <add name="Member" type="System.Boolean" />

</properties>
```

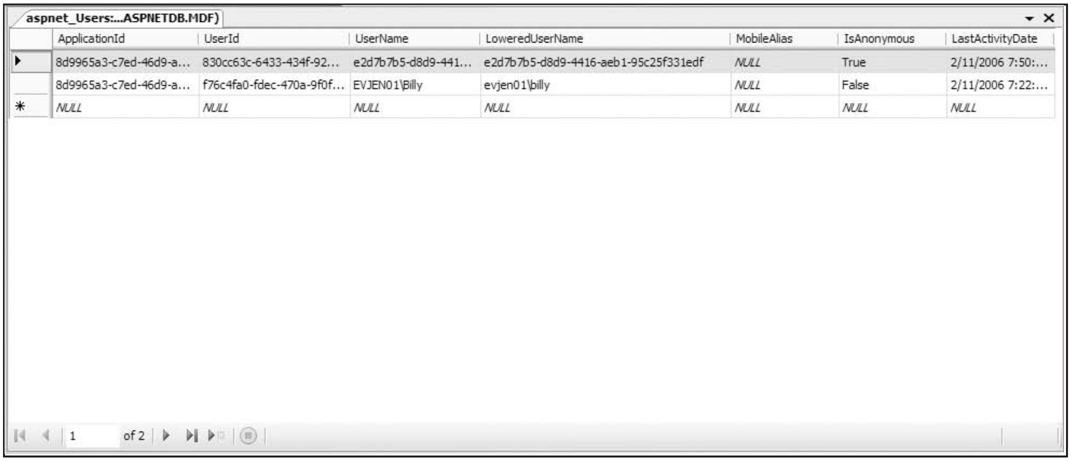
In this example, the `LastVisited` property is set to allow anonymous users by setting the `allowAnonymous` attribute to `true`. Because this is the only property that works with anonymous users, the rest of the defined properties do not store information for these types of users.

Warnings about Anonymous User Profile Storage

Taking into account everything said so far about anonymous users, you should be very careful about how you approach this option. Storing profile information about anonymous users can dramatically populate the data store you are using. For instance, in my examples, I am using Microsoft's SQL Server Express Edition, and I stored profile information for one authenticated user and then for a single anonymous user. This puts information for both these users in the aspnet_Profile and the aspnet_Users table.

The two users listed in the aspnet_Users table are shown in Figure 15-5.

In this figure, the anonymous user is highlighted with the gray bar, and you can see that this user has a pretty cryptic name, which is the `Request.AnonymousID` presented earlier. The other big difference between the two users is shown with the `IsAnonymous` column in the table. The anonymous user has a setting of `true` for this column while the authenticated user has a setting of `false`. Because your database



aspnet_Users:...ASPNETDB.MDF)

	ApplicationId	UserId	UserName	LoweredUserName	MobileAlias	IsAnonymous	LastActivityDate
▶	8d9965a3-c7ed-46d9-a...	830cc63c-6433-434f-92...	e2d7b7b5-d8d9-441...	e2d7b7b5-d8d9-4416-aeb1-95c25f331edf	NULL	True	2/11/2006 7:50:...
	8d9965a3-c7ed-46d9-a...	f76c4fa0-fdec-470a-9f0f...	EVJEN01\Billy	evjen01\billy	NULL	False	2/11/2006 7:22:...
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 15-5

can fill up quickly with anonymous user information, you should weigh which information you really need to store on these types of users.

Programmatic Access to Personalization

When an ASP.NET page is invoked, ASP.NET creates a class (`ProfileCommon`) by inheriting from the `ProfileBase` class, which it uses to strongly type the profile properties that were defined in the `web.config` file. This created class, meant to deal with the user's profile store, gets and sets profile properties through the use of the `GetPropertyValue` and `SetPropertyValue` methods from the `ProfileBase` class.

As you would expect, ASP.NET provides you with the hooks necessary to get at specific `Profile` events using the `ProfileModule` class. The `ProfileModule` class is what ASP.NET itself uses to create and store profile information in the page's `Profile` object.

The `ProfileModule` class exposes three events that you can use to handle your user's profile situations. These events, `MigrateAnonymous`, `Personalize`, and `ProfileAutoSaving`, are focused around the area of authentication. Because this section just showed you how to work with anonymous users in your applications, this section now looks at how to migrate these users from anonymous users to authenticated users — because you are most likely going to want to move their profile properties as well as change their status.

Migrating Anonymous Users

When working with anonymous users, you must be able to migrate anonymous users to registered users. For example, after an end user fills a shopping cart, he can register on the site to purchase the items. At that moment, the end user switches from being an anonymous user to a registered user.

For this reason, ASP.NET provides a `Profile_MigrateAnonymous` event handler enabling you to migrate anonymous users to registered users. The `Profile_MigrateAnonymous` event requires a data class of type `ProfileMigrateEventArgs`. It is placed either in the page that deals with the migration or within the `Global.asax` file (if it can be used from anywhere within the application). The use of this event is illustrated in Listing 15-17.

Listing 15-17: Migrating anonymous users for particular personalization properties

VB

```
Public Sub Profile_MigrateAnonymous(ByVal sender As Object, _
   ByVal e As ProfileMigrateEventArgs)

      Dim anonymousProfile As ProfileCommon = Profile.GetProfile(e.AnonymousID)
      Profile.LastVisited = anonymousProfile.LastVisited

End Sub
```

C#

```
public void Profile_MigrateAnonymous(object sender,
   ProfileMigrateEventArgs e)
{
      ProfileCommon anonymousProfile = Profile.GetProfile(e.AnonymousID);
      Profile.LastVisited = anonymousProfile.LastVisited
}
```

From this example, you create an instance of the `ProfileCommon` object and populate it with the profile from the visiting anonymous user. Then from there, you can use the instance to get at all the profile properties of that anonymous user. That means that you can then populate a profile through a movement from the anonymous user's profile information to the authenticated user's profile system.

Listing 15-17 shows how to migrate a single personalization property from an anonymous user to the new registered user. In addition to migrating single properties, you can also migrate properties that come from personalization groups. This is shown in Listing 15-18.

Listing 15-18: Migrating anonymous users for items in personalization groups

VB

```
Public Sub Profile_MigrateAnonymous(ByVal sender As Object, _
   ByVal e As ProfileMigrateEventArgs)

      Dim au As ProfileCommon = Profile.GetProfile(e.AnonymousID)

       If au.MemberDetails.DateJoined <> "" Then
          Profile.MemberDetails.DateJoined = DateTime.Now().ToString()
           Profile.FamilyDetails.MarriedStatus = au.FamilyDetails.MarriedStatus
       End If

      AnonymousIdentificationModule.ClearAnonymousIdentifier()
End Sub
```

C#

```
public void Profile_MigrateAnonymous(object sender,
   ProfileMigrateEventArgs e)
{
        ProfileCommon au = Profile.GetProfile(e.AnonymousID);

        if (au.MemberDetails.DateJoined != String.Empty) {
           Profile.MemberDetails.DateJoined = DateTime.Now.ToString();
           Profile.FamilyDetails.MarriedStatus = au.FamilyDetails.MarriedStatus;
        }

        AnonymousIdentificationModule.ClearAnonymousIdentifier();
   }
```

Using this event in the `Global.asax` file enables you to logically migrate anonymous users as they register themselves with your applications. The migration process also allows you to pick and choose which items you migrate and to change the values as you wish.

Personalizing Profiles

Besides working with anonymous users from the `Global.asax` file, you can also programmatically personalize the profiles retrieved from the personalization store. This is done through the use of the `Profile_Personalize` event. An example use of this event is shown in Listing 15-19.

Listing 15-19: Personalizing a retrieved profile

VB

```
Public Sub Profile_Personalize(sender As Object, args As ProfileEventArgs)
  Dim checkedProfile As ProfileCommon

  If User Is Nothing Then Return

  checkedProfile = CType(ProfileBase.Create(User.Identity.Name), ProfileCommon)

  If (Date.Now.IsDaylightSavingTime()) Then
    checkedProfile = checkedProfile.GetProfile("TimeDifferenceUser")
  Else
    checkedProfile = checkedProfile.GetProfile("TimeUser")
  End If

  If Not checkedProfile Is Nothing Then
    args.Profile = checkedProfile
  End If
End Sub
```

C#

```
public void Profile_Personalize(object sender, ProfileEventArgs args)
{
  ProfileCommon checkedProfile;

  if (User == null) { return; }

  checkedProfile = (ProfileCommon)ProfileBase.Create(User.Identity.Name);

  if (DateTime.Now.IsDaylightSavingTime()) {
    checkedProfile = checkedProfile.GetProfile("TimeDifferenceUser");
```

Continued

```
      }
      else {
        checkedProfile = checkedProfile.GetProfile("TimeUser");
      }

      if (checkedProfile != null) {
        args.Profile = checkedProfile;
      }
    }
```

In this case, based on a specific parameter (whether it is Daylight Savings Time or something else), you are able to assign a specific profile to the user. You do this by using the `ProfileModule.Personalize` event, which you would usually stick inside the `Global.asax` page.

Determining Whether to Continue with Automatic Saves

When you are working with the profile capabilities provided by ASP.NET, the page automatically saves the profile values to the specified data store at the end of the page's execution. This capability, which is turned on (set to `true`) by default, can be set to `false` through the use of the `automaticSaveEnabled` attribute in the `<profile>` node in the `web.config` file. This is illustrated in Listing 15-20.

Listing 15-20: Working with the automaticSaveEnabled attribute

```
<profile automaticSaveEnabled="false">

   <properties>

      <add name="FirstName" />
      <add name="LastName" />
      <add name="LastVisited" />
      <add name="Age" />
      <add name="Member" />

   </properties>

 </profile>
```

If you have set the `automaticSaveEnabled` attribute value to `false`, you will have to invoke the `ProfileBase.Save()` method yourself. In most cases though, you are going to leave this setting on `true`. Once a page request has been made and finalized, the `ProfileModule.ProfileAutoSaving` event is raised. This is an event that you can also work with, as shown in Listing 15-21.

Listing 15-21: Using the ProfileAutoSaving event to turn off the auto-saving feature

VB

```
Public Sub Profile_ProfileAutoSaving(sender As Object, _
 args As ProfileAutoSaveEventArgs)

  If Profile.PaidDueStatus.HasChanged Then
    args.ContinueWithProfileAutoSave = True
  Else
```

```
    args.ContinueWithProfileAutoSave = False
  End If
End Sub
```

C#

```
public void Profile_ProfileAutoSaving(object sender, ProfileAutoSaveEventArgs args)
{
  if (Profile.PaidDueStatus.HasChanged)
    args.ContinueWithProfileAutoSave = true;
  else
    args.ContinueWithProfileAutoSave = false;
}
```

In this case, when the `Profile_ProfileAutoSaving` event is triggered, it is then possible to work within this event and change some behaviors. Listing 15-21 looks to see if the `Profile.PaidDueStatus` property has changed. If it has changed, the auto-saving feature of the profile system is continued; if the `Profile.PaidDueStatus` has not changed, the auto-saving feature is turned off.

Personalization Providers

As shown in Figure 15-1 earlier in the chapter, the middle tier of the personalization model, the personalization API layer, communicates with a series of default data providers. By default, the personalization model uses Microsoft SQL Server Express Edition files for storing the personalization properties you define. You are not limited to just this type of data store, however. You can also use the Microsoft SQL Server data provider to allow you to work with Microsoft SQL Server 7.0, 2000, 2005, and SQL Server 2008. Besides the Microsoft SQL Server data provider, the architecture also allows you to create your own data providers if one of these data stores does not fit your requirements.

Working with SQL Server Express Edition

The Microsoft SQL Server data provider does allow you to work with the new SQL Server Express Edition files. The SQL Server data provider is the default provider used by the personalization system provided by ASP.NET. When used with Visual Studio 2008, the IDE places the `ASPNETDB.MDF` file within your application's `App_Data` folder.

As you look through the `machine.config` file, notice the sections that deal with how the personalization engine works with this database. In the first reference to the `LocalSqlServer` file, you find a connection string to this file (shown in Listing 15-22) within the `<connectionStrings>` section of the file.

Listing 15-22: Adding a connection string to the SQL Server Express file

```
<configuration>

    <connectionStrings>
        <clear />
        <add name="LocalSqlServer"
         connectionString="data source=.\SQLEXPRESS;Integrated Security=SSPI;
         AttachDBFilename=|DataDirectory|aspnetdb.mdf;User Instance=true"
         providerName="System.Data.SqlClient" />
    </connectionStrings>

</configuration>
```

In this example, you see that a connection string with the name `LocalSqlServer` has been defined. The location of the file, specified by the `connectionString` attribute, points to the relative path of the file. This means that in every application you build that utilizes the personalization capabilities, the default SQL Server provider should be located in the application's `App_Data` folder and have the name of `ASPNETDB.MDF`.

The SQL Server Express file's connection string is specified through the `LocalSqlServer` declaration within this `<connectionStrings>` section. You can see the personalization engine's reference to this in the `<profile>` section within the `machine.config` file. The `<profile>` section includes a subsection listing all the providers available to the personalization engine. This is shown in Listing 15-23.

Listing 15-23: Adding a new SQL Server data provider

```
<configuration>
  <system.web>

        <profile>
            <providers>
                <add name="AspNetSqlProfileProvider"
                 connectionStringName="LocalSqlServer" applicationName="/"
                 type="System.Web.Profile.SqlProfileProvider, System.Web,
                   Version=2.0.0.0, Culture=neutral,
                   PublicKeyToken=b03f5f7f11d50a3a" />
            </providers>
        </profile>

  </system.web>
</configuration>
```

From this, you can see that a provider is added by using the `<add>` element. Within this element, the `connectionStringName` attribute points to what was declared in the `<connectionString>` attribute from Listing 15-22.

You can specify an entirely different Microsoft SQL Server Express Edition file other than the one specified in the `machine.config` file. First, create a connection string that points to a new SQL Server Express file that is a templated version of the `ASPNETDB.mdb` file. At this point, you can use `<connectionString>` to point to this new file. If you change these values in the `machine.config` file, all the ASP.NET applications that reside on the server will then use this specified file. If you make the changes only to the `web.config` file, however, only the application using this particular `web.config` file uses this new data store. Other applications on the server remain unchanged.

Working with Microsoft's SQL Server 7.0/2000/2005/2008

You will likely find it quite easy to work with the personalization framework using the SQL Server Express files. But when you work with larger applications that require the factors of performance and reliability, you should use the SQL Server personalization provider along with SQL Server 7.0, 2000, 2005, or 2008. If this data store is available, you should always try to use this option instead of the default SQL Server Express Edition files.

If you worked with the SQL Server personalization provider using SQL Server Express files as explained earlier, you probably found it easy to use. The personalization provider works right out of the box — without any set up or configuration on your part. Using the SQL Server personalization provider with a full-blown version of SQL Server, however, is a bit of a different story. Although it is not difficult to work with, you must set up and configure your SQL Server before using it.

ASP.NET 3.5 provides a couple of ways to set up and configure SQL Server for the personalization framework. One way is through the ASP.NET SQL Server Setup Wizard, and the other method is by running some of the SQL Server scripts provided with the .NET Framework 2.0.

Using the ASP.NET SQL Server Setup Wizard is covered in detail in Chapter 12.

To use this wizard to set up your SQL Server for the ASP.NET 3.5 personalization features, you must first open up the aspnet_regsql.exe tool by invoking it from the Visual Studio 2008 Command Prompt. You open this command prompt by selecting Start ➪ All Programs ➪ Visual Studio 2008 ➪ Visual Studio Tools ➪ Visual Studio 2008 Command Prompt. At the prompt, type in `aspnet_regsql.exe` to open the GUI of the ASP.NET SQL Server Setup Wizard. If you step through the wizard, you can set up your SQL Server instance for many of the ASP.NET systems, such as the personalization system.

Using SQL Scripts to Install Personalization Features

Another option is to use the same SQL scripts that these tools and wizards use. If you look at `C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\`, from this location, you can see the install and remove scripts — `InstallPersonalization.sql` and `UninstallPersonalization.sql`. Running these scripts provides your database with the tables needed to run the personalization framework. Be forewarned that you must run the `InstallCommon.sql` script before running the personalization script (or any of the new other ASP.NET system scripts).

Configuring the Provider for SQL Server 2005

After you have set up your SQL Server database for the personalization system, the next step is to redefine the personalization provider so that it works with this instance (instead of with the default Microsoft SQL Server Express Edition files).

You accomplish this in the `web.config` file of your application. Here, you want to configure the provider and then define this provider instance as the provider to use. Listing 15-24 shows these additions plus the enlarged `<profile>` section of the `web.config` file.

Listing 15-24: Connecting the SqlProfileProvider to SQL Server 2005

```
<configuration>

     <connectionStrings>
        <add name="LocalSql2005Server"
         connectionString="data source=127.0.0.1;Integrated Security=SSPI" />
     </connectionStrings>

     <profile defaultProvider="AspNetSql2005ProfileProvider">
        <providers>
```

Continued

```
            <clear />
            <add name="AspNetSql2005ProfileProvider"
             connectionStringName="LocalSql2005Server" applicationName="/"
             type="System.Web.Profile.SqlProfileProvider, System.Web,
               Version=2.0.0.0, Culture=neutral,
               PublicKeyToken=b03f5f7f11d50a3a" />
         </providers>

         <properties>
            <add name="FirstName" />
            <add name="LastName" />
            <add name="LastVisited" />
            <add name="Age" />
            <add name="Member" />
         </properties>
      </profile>

</configuration>
```

The big change you make to this profile definition is to use the `defaultProvider` attribute with a value that is the name of the provider you want to use — in this case the newly created SQL Server provider, `AspNetSql2005ProfileProvider`. You can also make this change to the `machine.config` file by changing the `<profile>` element, as shown in Listing 15-25.

Listing 15-25: Using SQL Server as the provider in the machine.config file

```
<configuration>
  <system.web>

  ...

     <profile enabled="true" defaultProvider="AspNetSql2005ProfileProvider">

       ...

      </profile>

   ...

  </system.web>
</configuration>
```

This change forces each and every application that resides on this server to use this new SQL Server provider instead of the default SQL Server provider (unless this command is overridden in the application's `web.config` file).

Using Multiple Providers

You are not limited to using a single data store or provider. Instead, you can use any number of providers. You can even specify the personalization provider for each property defined. This means that you can use the default provider for most properties, as well as allowing a few of them to use an entirely different provider (see Listing 15-26).

Listing 15-26: Using different providers

```
<configuration>
  <system.web>

     <profile
      defaultProvider="AspNetSqlProvider">

                <properties>

                     <add name="FirstName" />
                     <add name="LastName" />
                     <add name="LastVisited" />
                     <add name="Age" />
                     <add name="Member" provider="AspNetSql2005ProfileProvider" />

                </properties>

       </profile>

  </system.web>
</configuration>
```

From this example, you can see that a default provider is specified — `AspNetSqlProvider`. Unless specified otherwise, this provider is used. The only property that changes this setting is the property `Member`. The `Member` property uses an entirely different personalization provider. In this case, it employs the Access provider (`AspNetSql2005ProfileProvider`) through the use of the `provider` attribute of the `<add>` element. With this attribute, you can define a specific provider for each and every property that is defined.

Managing Application Profiles

When you put into production an ASP.NET application that uses profile information, you quickly realize that you need a way to manage all the profile information collected over the lifecycle of the application. As you look at the ASP.NET MMC snap-in or the ASP.NET Web Site Administration Tool, note that neither of these tools gives you a way to delete a specific user's profile information or even to cleanse a database of profile information for users who haven't been active in awhile.

ASP.NET 3.5 gives you the means to manage the profile information that your application stores. This is done through the use of the `ProfileManager` class available in .NET.

Through the use of the `ProfileManager` class, you can build-in the administration capabilities to completely manage the profile information that is stored by your application. In addition to being able to access property values, such as the name of the provider being used by the personalization system or the name of the application in question, you also have a large number of methods available in the `ProfileManager` class to retrieve all sorts of other information concerning your user's profile. Through the `ProfileManager` class, you also have the capability to perform actions on this stored profile information including cleansing the database of old profile information.

Properties of the ProfileManger Class

The properties of the `ProfileManager` class are detailed in the following table.

Properties	**Description**
`ApplicationName`	Gets or sets the name of the application to work with.
`AutomaticSaveEnabled`	Gets or sets a Boolean value indicating whether the profile information is stored at the end of the page execution.
`Enabled`	Gets or sets a Boolean value indicating whether the application is able to use the personalization system.
`Provider`	Gets the name of the provider being used for the personalization system.
`Providers`	Gets a collection of all the providers available for the ASP.NET application.

You can see that these properties include a bit of information about the personalization system and the providers available to it that you can integrate into any management system you build. Next, this chapter looks at the methods available for the `ProfileManager` class.

Methods of the ProfileManager Class

A good number of methods are available to the `ProfileManager` class that help you manage the profiles of the users of your application. These methods are briefly described in the following table.

Properties	**Description**
`DeleteInactiveProfiles`	Provides you with the capability to delete any profiles that haven't seen any activity for a specified time period.
`DeleteProfile`	Provides you with the capability to delete a specific profile.
`DeleteProfiles`	Provides you with the capability to delete a collection of profiles.
`FindInactive Profiles ByUserName`	Provides you with all the inactive profiles under a specific username according to a specified date.
`FindProfilesBy UserName`	Provides you with all the profiles from a specific username.
`GetAllInactiveProfiles`	Provides you with all the profiles that have been inactive since a specified date.
`GetAllProfiles`	Provides you with a collection of all the profiles.
`GetNumberOf InactiveProfiles`	Provides you with the number of inactive profiles from a specified date.
`GetNumberOfProfiles`	Provides you with the number of total profiles in the system.

As you can see from this list of methods, you can do plenty to manage to the profile information that is stored in your database.

Next, this chapter looks at building a profile manager administration page for your ASP.NET application. This example builds it as an ASP.NET page, but you can just as easily build it as a console application.

Building the ProfileManager.aspx Page

To create a simple profile manager for your application, create a single ASP.NET page in your application called `ProfileManager.aspx`. You use this page to manage the profiles that are stored in the database for this particular application.

This page includes a number of controls, but the most important is a DropDownList control that holds all the usernames of entities that have profile information in the database. You might see the same username a couple of times depending on what you are doing with your application. Remember that a single user can have multiple profiles in the database.

Using the DropDownList control, you can select a user and see information about his profile stored in your data store. From this page, you can also delete his profile information. You can actually perform very many operations with the `ProfileManager` class, but this is a good example of some basic ones.

The code for the `ProfileManager.aspx` page is presented in Listing 15-27.

Listing 15-27: The ProfileManager.aspx page

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        If (DropDownList1.Items.Count = 0) Then
            WriteDropdownList()
            WriteUserOutput()
        End If
    End Sub

    Protected Sub DeleteButton_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        ProfileManager.DeleteProfile(DropDownList1.Text.ToString())
        DropDownList1.Items.Clear()
        WriteDropdownList()
        WriteUserOutput()
    End Sub

    Protected Sub SelectButton_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        WriteUserOutput()
    End Sub

    Protected Sub WriteUserOutput()
        Dim outputInt As Integer
```

Continued

```
        Dim pic As ProfileInfoCollection = New ProfileInfoCollection()
        pic = ProfileManager.
          FindProfilesByUserName(ProfileAuthenticationOption.All, _
          DropDownList1.Text.ToString(), 0, 1, outputInt)

        DetailsView1.DataSource = pic
        DetailsView1.DataBind()
    End Sub

    Protected Sub WriteDropdownList()
        Dim outputInt As Integer
        Dim pic As ProfileInfoCollection = New ProfileInfoCollection()
        pic = ProfileManager.Provider.
           GetAllProfiles(ProfileAuthenticationOption.All, 0, 10000, outputInt)

        For Each proInfo As ProfileInfo In pic
            Dim li As ListItem = New ListItem()
            li.Text = proInfo.UserName.ToString()

            DropDownList1.Items.Add(li)
        Next

        Label1.Text = outputInt.ToString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>ProfileAdmin Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <b>Profile Manager<br />
        </b>
        <br />
        Total number of users in system:
        <asp:Label ID="Label1" runat="server"></asp:Label><br />
        &nbsp;<br />
        <asp:DropDownList ID="DropDownList1" runat="server">
        </asp:DropDownList>&nbsp;
        <asp:Button ID="SelectButton" runat="server"
         OnClick="SelectButton_Click"
            Text="Get User Profile Information" /><br />
        <br />
        <asp:DetailsView ID="DetailsView1" runat="server" CellPadding="4"
         ForeColor="#333333" GridLines="None"
            Height="50px">
             <FooterStyle BackColor="#1C5E55" Font-Bold="True" ForeColor="White" />
             <EditRowStyle BackColor="#7C6F57" />
            <PagerStyle BackColor="#666666" ForeColor="White"
             HorizontalAlign="Center" />
            <HeaderStyle BackColor="#1C5E55" Font-Bold="True" ForeColor="White" />
            <AlternatingRowStyle BackColor="White" />
```

Continued

```
            <CommandRowStyle BackColor="#C5BBAF" Font-Bold="True" />
            <RowStyle BackColor="#E3EAEB" />
            <FieldHeaderStyle BackColor="#D0D0D0" Font-Bold="True" />
        </asp:DetailsView>
        <br />
        <asp:Button ID="DeleteButton" runat="server"
         Text="Delete Selected User's Profile Information"
         OnClick="DeleteButton_Click" />
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        if (DropDownList1.Items.Count == 0)
        {
            WriteDropdownList();
            WriteUserOutput();
        }
    }

    protected void DeleteButton_Click(object sender, EventArgs e)
    {
        ProfileManager.DeleteProfile(DropDownList1.Text.ToString());
        DropDownList1.Items.Clear();
        WriteDropdownList();
        WriteUserOutput();
    }

    protected void SelectButton_Click(object sender, EventArgs e)
    {
        WriteUserOutput();
    }

    protected void WriteUserOutput()
    {
        int outputInt;
        ProfileInfoCollection pic = new ProfileInfoCollection();
        pic = ProfileManager.FindProfilesByUserName
           (ProfileAuthenticationOption.All,
            DropDownList1.Text.ToString(), 0, 1, out outputInt);

        DetailsView1.DataSource = pic;
        DetailsView1.DataBind();
    }

    protected void WriteDropdownList()
```

Continued

```
        {
            int outputInt;
            ProfileInfoCollection pic = ProfileManager.Provider.GetAllProfiles
               (ProfileAuthenticationOption.All, 0, 10000, out outputInt);

            foreach (ProfileInfo proInfo in pic)
            {
                ListItem li = new ListItem();
                li.Text = proInfo.UserName.ToString();

                DropDownList1.Items.Add(li);
            }

            Label1.Text = outputInt.ToString();
        }
    </script>
```

Examining the Code of ProfileManager.aspx Page

As you look over the code of the `ProfileManager.aspx` page, note that the `ProfileManager` class is used to perform a couple of different operations.

First, the `ProfileManager` class's `GetAllProfiles()` method is used to populate the DropDownList control that is on the page. The constructor of this method is presented here:

```
GetAllProfiles(
   authenticationOption,
   pageIndex,
   pageSize,
   totalRecords)
```

The `GetAllProfiles()` method takes a number of parameters, the first of which allows you to define whether you are using this method for *all* profiles in the system, or just the anonymous or authenticated user's profiles contained in the system. In the case of this example, all the profiles are retrieved with this method. This is accomplished using the `ProfileAuthenticationOption` enumeration. Then, the other parameters of the `GetAllProfiles()` method require you to specify a page index and the number of records to retrieve from the database. There is not a *get all* option (because of the potential size of the data that might be retrieved); so instead, in this example, I specify the first page of data (using `0`) and that this page contains the first 10,000 records (which is basically a *get all* for my application). The last parameter of the `GetAllProfiles()` method enables you to retrieve the count of the records if you want to use that anywhere within your application or if you want to use that number to iterate through the records. The `ProfileManager.aspx` page uses this number to display within the `Label1` server control.

In return from the `GetAllProfiles()` method, you get a `ProfileInfoCollection` object, which is a collection of `ProfileInfo` objects. Iterating through all the `ProfileInfo` objects in the `ProfileInfoCollection`, you are able to pull out the some of the main properties for a particular user's profile information. In this example, just the `UserName` property of the `ProfileInfo` object is used to populate the DropDownList control on the page.

When the end user selects one of the users from the dropdown list, the `FindProfilesByUserName()` method is used to display the profile of the selected user. Again, a `ProfileInfoCollection` object is returned from this method as well.

To delete the profile of the user selected in the DropDownList control, simply use the `DeleteProfile()` method and pass in the name of the selected user as illustrated here:

```
ProfileManager.DeleteProfile(DropDownList1.Text.ToString())
DropDownList1.Items.Clear()
WriteDropdownList()
WriteUserOutput()
```

After the profile is deleted from the system, that name will not appear in the drop-down list anymore (because the DropDownList control has been redrawn). If you look in the database, particularly at the aspnet_Profile table, you see that the profile of the selected user is, in fact, deleted. However, also notice that the user (even if the user is anonymous) is still stored in the aspnet_Users table.

If you want to delete not only the profile information of the user but also delete the user from the aspnet_Users table, you should invoke the `DeleteUser()` method from the `Membership` class as presented here.

```
ProfileManager.DeleteProfile(DropDownList1.Text.ToString())
Membership.DeleteUser(DropDownList1.Text.ToString())
DropDownList1.Items.Clear()
WriteDropdownList()
WriteUserOutput()
```

This use of the `DeleteUser()` method also deletes the selected user from the aspnet_Users table. You could have also achieved the same thing by using the other constructor of the `DeleteUser()` method as presented here:

```
Membership.DeleteUser(DropDownList1.Text.ToString(), True)
DropDownList1.Items.Clear()
WriteDropdownList()
WriteUserOutput()
```

The second parameter used in this operation of the `DeleteUser()` method deletes all data related to that user across *all* the tables held in the `ASPNETDB.mdf` database.

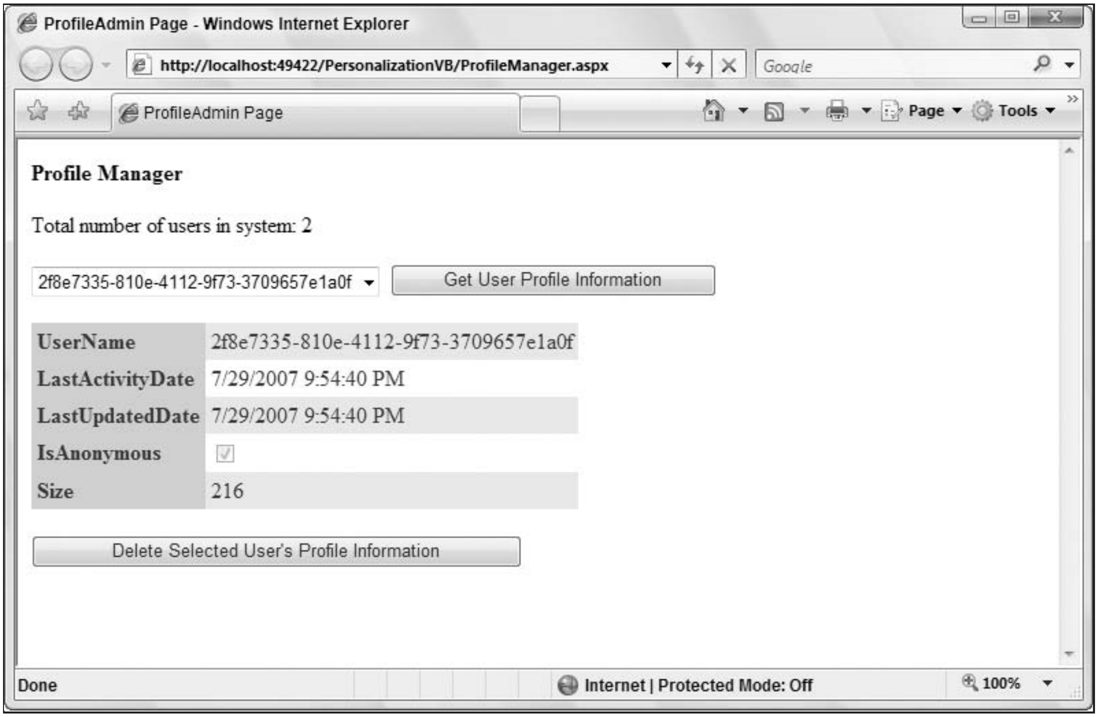
Running the ProfileManager.aspx Page

When you compile and run this page, you see results similar to those shown in Figure 15-6.

From this screen, you can see that this page is dealing with an anonymous user (based upon the GUID for the username). You can also see that the `IsAnonymous` column is indeed checked. From this page, you can then delete this user's profile information by selecting the appropriate button on the page.

Summary

The personalization capabilities provided by ASP.NET 3.5 make it incredibly easy to make your Web applications unique for all end users, whether they are authenticated or anonymous. This system enables you to store everything from basic data types provided by the .NET Framework to custom types that you create. This system is more versatile and extensible than using the `Session` or `Application` objects. The data is stored via a couple of built-in personalization providers that ship with ASP.NET. These providers


ProfileAdmin Page - Windows Internet Explorer
http://localhost:49422/PersonalizationVB/ProfileManager.aspx
Google
ProfileAdmin Page
Page
Tools
Get User Profile Information
Delete Selected User's Profile Information
Done
Internet | Protected Mode: Off
100%

Profile Manager

Total number of users in system: 2

2f8e7335-810e-4112-9f73-3709657e1a0f

UserName	2f8e7335-810e-4112-9f73-3709657e1a0f
LastActivityDate	7/29/2007 9:54:40 PM
LastUpdatedDate	7/29/2007 9:54:40 PM
IsAnonymous	☑
Size	216

Figure 15-6

include ones that connect with either Microsoft's SQL Server Express Edition files or Microsoft SQL Server 2008, 2005, 2000, or 7.0.

You can also use the `ProfileManager` class to manage your system's profile information. This includes the capability to monitor and delete profiles as you deem necessary.

16

Membership and Role Management

The authentication and authorization of users are important functions in many Web sites and browser-based applications. Traditionally, when working with Microsoft's Windows Forms applications (thick-client), you depended on Windows Integrated Authentication; when working with browser-based applications (thin-client), you used forms authentication.

Forms authentication enabled you to take requests that were not yet authenticated and redirect them to an HTML form using HTTP client-side redirection. The user provided his login information and submitted the form. After the application authenticated the request, the user received an HTTP cookie, which was then used on any subsequent requests. This kind of authentication was fine in many ways, but it required developers to build every element and even manage the back-end mechanics of the overall system. This was a daunting task for many developers and, in most cases, it was rather time-consuming.

ASP.NET 3.5 includes an authentication and authorization management service that takes care of the login, authentication, authorization, and management of users who require access to your Web pages or applications. This outstanding *membership and role management service* is an easy-to-implement framework that works out of the box using Microsoft SQL Server as the back-end data store. This framework also includes an API that allows for programmatic access to the capabilities of both the membership and role management services. In addition, a number of membership and role management–focused server controls make it easy to create Web applications that incorporate everything these services have to offer.

Before you look at the membership and role management features of ASP.NET 3.5, here's a quick review of authentication and authorization. This is vital to understand before proceeding.

Authentication

Authentication is a process that determines the identity of a user. After a user has been authenticated, a developer can determine if the identified user has *authorization* to proceed. It is impossible to give an entity authorization if no authentication process has been applied. Authentication is provided in ASP.NET 3.5 using the membership service.

Authorization

Authorization is the process of determining whether an authenticated user is allowed access to any part of an application, access to specific points of an application, or access only to specific datasets that the application provides. When you authenticate and authorize users or groups, you can customize a site based on user types or preferences. Authorization is provided in ASP.NET 3.5 using a role management service.

ASP.NET 3.5 Authentication

ASP.NET 3.5 provides the membership management service to deal with authenticating users to access a page or an entire site. The ASP.NET management service not only provides an API suite for managing users, but it also gives you some server controls, which in turn work with this API. These server controls work with the end user through the process of authentication. You look at the functionality of these controls shortly.

Setting Up Your Web Site for Membership

Before you can use the security controls that are provided with ASP.NET 3.5, you first have to set up your application to work with the membership service. How you do this depends on how you approach the security framework provided.

By default, ASP.NET 3.5 uses the built-in `SqlMembershipProvider` instance for storing details about the registered users of your application. For the initial demonstrations, the examples in this chapter work with forms-based authentication. You can assume for these examples that the application is on the public Internet and, therefore, is open to the public for registration and viewing. If it were an intranet-based application (meaning that all the users are on a private network), you could use Windows Integrated Authentication for authenticating users.

ASP.NET 3.5, as you know, offers a data provider model that handles the detailed management required to interact with multiple types of underlying data stores. Figure 16-1 shows a diagram of the ASP.NET 3.5 membership service.

From the diagram, you can see that, like the rest of the ASP.NET provider models, the membership providers can access a wide variety of underlying data stores. In this diagram, you can see the built-in Microsoft SQL Server data store. You can also build your own membership providers to get at any other custom data stores that work with user credentials. Above the membership providers in the diagram, you can see a collection of security-focused server controls that utilize the access granted by the underlying membership providers to work with the users in the authentication process.

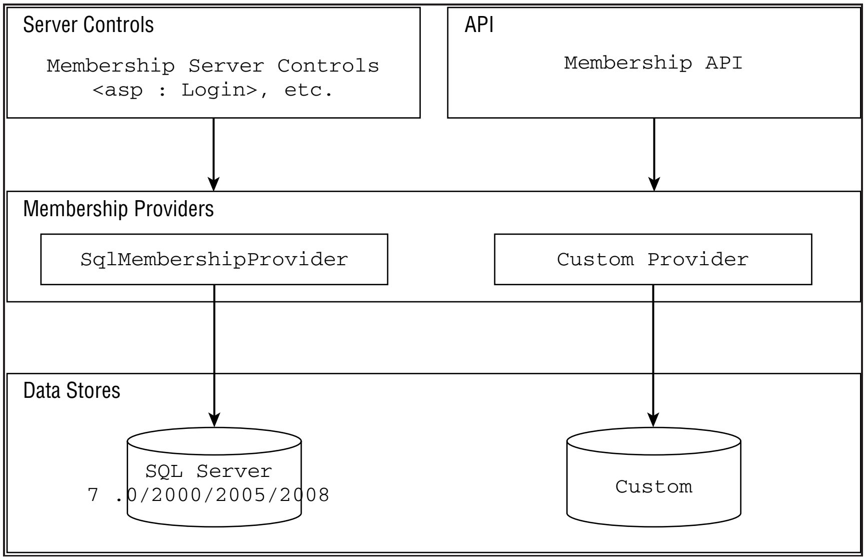

Server Controls
Membership Server Controls
<asp : Login>, etc.
API
Membership API
Membership Providers
SqlMembershipProvider
Custom Provider
Data Stores
SQL Server
7 .0/2000/2005/2008
Custom

Figure 16-1

Adding an <authentication> Element to the web.config File

In order to have the forms authentication element in your Web application work with the membership service, the first step is to turn on forms authentication within the `web.config` file. To accomplish this, create a `web.config` file (if you do not already have one in your application). Next, add the section shown in Listing 16-1 to this file.

Listing 16-1: Adding forms authentication to the web.config file

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <system.web>
        <authentication mode="Forms" />
    </system.web>
</configuration>
```

The simple addition of the `<authentication>` element to the `web.config` file turns on everything that you need to start using the membership service provided by ASP.NET 3.5. To turn on the forms authentication using this element, you simply give the value `Forms` to the `mode` attribute. This is a forms authentication example, but other possible values of the `mode` attribute include `Windows`, `Passport`, or `None`.

IIS authentication schemes include basic, digest, and Integrated Windows Authentication. Passport authentication points to a centralized service provided by Microsoft that offers a single login and core profile service for any member sites. It costs money to use Passport, which has also recently been depreciated by Microsoft.

Because the `mode` attribute in our example is set to `Forms`, you can move on to the next step of adding users to the data store. You can also change the behavior of the forms authentication system at this point by making some modifications to the `web.config` file. These possibilities are reviewed next.

Adding a <forms> Element to the web.config File

Using forms authentication, you can provide users with access to a site or materials based upon credentials they input into a Web-based form. When an end user attempts to access a Web site, he is entering the site using anonymous authentication, which is the default authentication mode. If he is found to be anonymous, he can be redirected (by ASP.NET) to a specified login page. After the end user inputs the appropriate login information and passes the authentication process, he is provided with an HTTP cookie, which can be used in any subsequent requests.

You can modify the behavior of the forms-based authentication by defining that behavior within a `<forms>` section in the `web.config` file. You can see the possibilities of the forms authentication setting in Listing 16-2, which shows possible changes to the `<forms>` section in the `web.config` file.

Listing 16-2: Modifying the forms authentication behavior

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <system.web>
        <authentication mode="Forms">
           <forms name=".ASPXAUTH"
                  loginUrl="Login.aspx"
                  protection="All"
                  timeout="30"
                  path="/"
                  requireSSL="false"
                  slidingExpiration="true"
                  cookieless="UseDeviceProfile" />
        </authentication>
    </system.web>
</configuration>
```

You can set these as you wish, and you have plenty of options for values other than the ones that are displayed. Also, as stated earlier, these values are not required. You can use the membership service right away with only the configuration setting that is shown in Listing 16-1.

You can find some interesting settings in Listing 16-2, however. You can really change the behavior of the forms authentication system by adding this `<forms>` element to the `web.config` file. If you do this, however, make sure that you have the `<forms>` element nested within the `<authentication>` elements. The following list describes the possible attributes of the `<forms>` element:

- ❑ `name`: Defines the name used for the cookie sent to end users after they have been authenticated. By default, this cookie is named `.ASPXAUTH`.
- ❑ `loginUrl`: Specifies the page location to which the HTTP request is redirected for logging in the user if no valid authentication cookie (`.ASPXAUTH` or otherwise) is found. By default, it is set to `Login.aspx`.
- ❑ `protection`: Specifies the amount of protection that you want to apply to the cookie that is stored on the end user's machine after he has been authenticated. The possible settings include `All`, `None`, `Encryption`, and `Validation`. You should always attempt to use `All`.
- ❑ `timeout`: Defines the amount of time (in minutes) after which the cookie expires. The default value is 30 minutes.

- ❑ `path`: Specifies the path for cookies issued by the application.
- ❑ `requireSSL`: Defines whether you require that credentials be sent over an encrypted wire (SSL) instead of clear text.
- ❑ `slidingExpiration`: Specifies whether the timeout of the cookie is on a sliding scale. The default value is `true`. This means that the end user's cookie does not expire until 30 minutes (or the time specified in the `timeout` attribute) after the last request to the application has been made. If the value of the `slidingExpiration` attribute is set to `false`, the cookie expires 30 minutes from the first request.
- ❑ `cookieless`: Specifies how the cookies are handled by ASP.NET. The possible values include `UseDeviceProfile`, `UseCookies`, `AutoDetect`, and `UseUri`. The default value is `UseDeviceProfile`. This value detects whether to use cookies based on the user agent of the device. `UseCookies` requires that all requests have the credentials stored in a cookie. `AutoDetect` auto-determines whether the details are stored in a cookie on the client or within the URI (this is done by sending a test cookie first). Finally, `UseUri` forces ASP.NET to store the details within the URI on all instances.

Now that forms authentication is turned on, the next step is adding users to the Microsoft SQL Server Express Edition data store, `ASPNETDB.mdf`.

Adding Users

To add users to the membership service, you can register users into the Microsoft SQL Server Express Edition data store. The first question you might ask is, ''Where is this data store?''

> *Of course, there are a number of editions of Microsoft's SQL Server that you can use to work through the examples in this book. With that said, this chapter uses the default database the membership system uses in creating users.*

The Microsoft SQL Server provider for the membership system can use a SQL Server Express Edition file that is structured specifically for the membership service (and other ASP.NET systems, such as the role management system). ASP.NET is set to automatically create this particular file for you if the appropriate file does not exist already. To create the `ASPNETDB.mdf` file, you work with the ASP.NET server controls that utilize an aspect of the membership service. When the application requires the `ASPNETDB.mdf` file, ASP.NET creates this file on your behalf in the `App_Data` folder.

After the data store is in place, it is time to start adding users to the data store.

Using the CreateUserWizard Server Control

The CreateUserWizard server control is one that can be used in conjunction with the membership service. You can find this and the other controls mentioned in this chapter under the Login section in the Visual Studio 2008 Toolbox. The CreateUserWizard control enables you to plug registered users into your data store for later retrieval. If a page in your application allows end users to register for your site, you want, at a minimum, to retrieve a login and password from the user and place these values in the data store. This enables the end user to access these items later to log in to the application using the membership system.

To make your life as simple as possible, the CreateUserWizard control takes complete control of registration on your behalf. Listing 16-3 shows a simple use of the control.

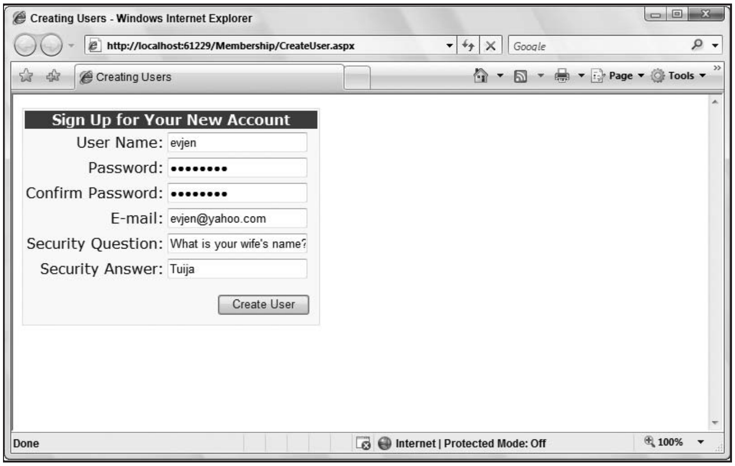
Listing 16-3: Allowing end users to register with the site

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Creating Users</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:CreateUserWizard ID="CreateUserWizard1" Runat="server"
         BorderWidth="1px" BorderColor="#FFDFAD" BorderStyle="Solid"
         BackColor="#FFFBD6" Font-Names="Verdana">
            <TitleTextStyle Font-Bold="True" BackColor="#990000"
             ForeColor="White"></TitleTextStyle>
        </asp:CreateUserWizard>
    </form>
</body>
</html>
```

This page simply uses the CreateUserWizard control and nothing more. This one control enables you to register end users for your Web application. This particular CreateUserWizard control has a little style applied to it, but this control can be as simple as:

```
<asp:CreateUserWizard ID="CreateUserWizard1" Runat="server">
</asp:CreateUserWizard>
```

When this code is run, an end user is presented with the form shown in Figure 16-2.


Creating Users - Windows Internet Explorer
http://localhost:61229/Membership/CreateUser.aspx
Google
Creating Users
Page
Tools
Create User
Done
Internet | Protected Mode: Off
100%

Sign Up for Your New Account

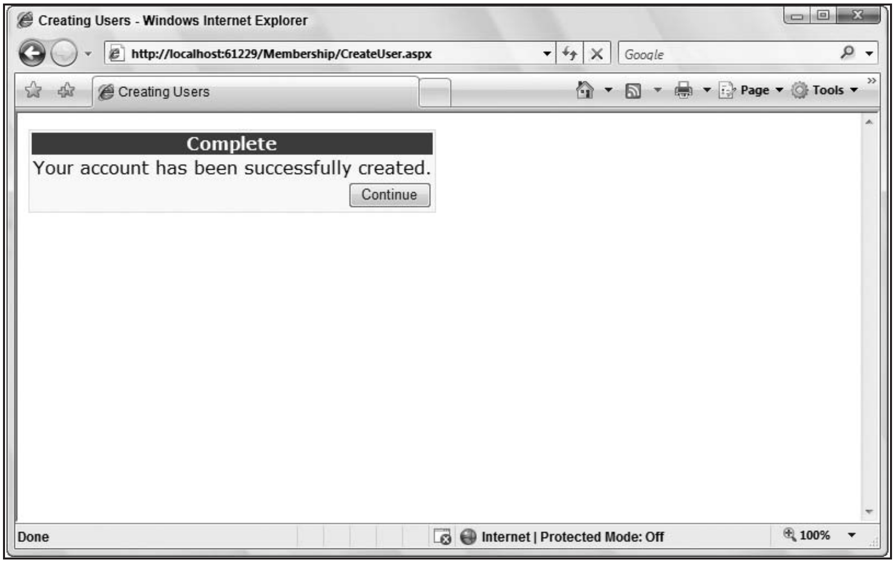
User Name: evjen
Password: ••••••••
Confirm Password: ••••••••
E-mail: evjen@yahoo.com
Security Question: What is your wife's name?
Security Answer: Tuija

Figure 16-2

This screenshot shows the form as it would appear when filled out by the end user and includes information such as the username, password, e-mail address, as well as a security question-and-answer section. Clicking the Create User button places this defined user information into the data store.

The username and password provided via this control enable the end user to log in to the application later through the Login server control. A Confirm Password text box is also included in the form of the CreateUser server control to ensure that the password provided is spelled correctly. An e-mail address text box is included (in case end users forget their login credentials and want the credentials e-mailed to them at some later point in time). Then finally, the security question and answer are used to verify the identity of the end user before any credentials or user information is changed or later provided via the browser.

After the Create User button on this form is clicked, the end user is presented with a confirmation of the information being stored (see Figure 16-3).



Creating Users - Windows Internet Explorer
http://localhost:61229/Membership/CreateUser.aspx
Google
Creating Users
Page
Tools
Done
Internet | Protected Mode: Off
100%

Complete

Your account has been successfully created.

Continue

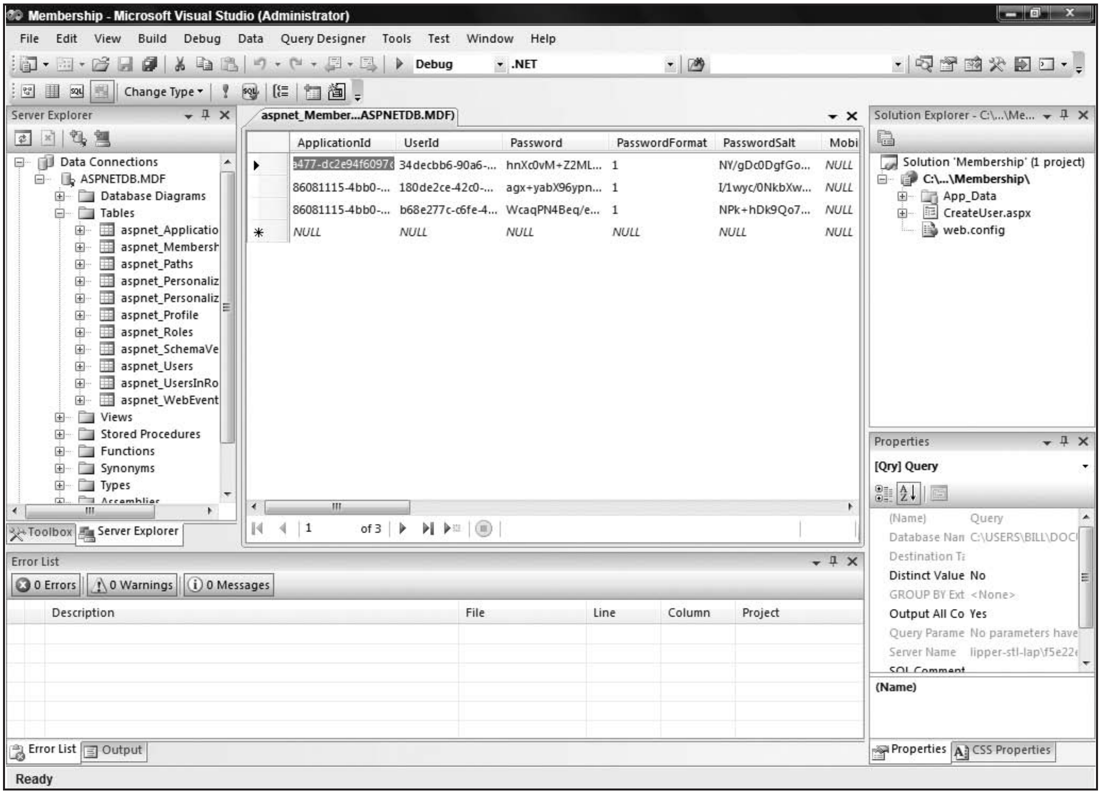
Figure 16-3

Seeing Where Users Are Stored

Now that the CreateUserWizard control has been used to add a user to the membership service, look at where this information is stored. If you used Visual Studio to create the Microsoft SQL Server Express Edition file in which you want to store the user information, the file is created when the previous example is run and you complete the form process as shown in the preceding figures. When the example is run and completed, you can click the Refresh button in the Solution Explorer to find the `ASPNETDB.mdf` file, which is located in the `App_Data` folder of your project. Many different tables are included in this file, but you are interested in the `aspnet_Membership` table only.

When you open the `aspnet_Membership` table (by right-clicking the table in the Server Explorer and selecting Show Table Data), the users you entered are in the system. This is illustrated in Figure 16-4.

The user password in this table is not stored as clear text; instead, it is hashed, which is a one way form of encryption that cannot be reversed easily. When a user logs in to an application that is using the ASP.NET 3.5 membership service, his or her password is immediately hashed and then compared to the hashed password stored in the database. If the two hashed strings do not compare, the passwords are not considered a match. Storing clear text passwords is considered a security risk, so you should never do so without weighing the risk involved.


Membership - Microsoft Visual Studio (Administrator)
File Edit View Build Debug Data Query Designer Tools Test Window Help
Debug
.NET
Change Type
Server Explorer
Data Connections
ASPNETDB.MDF
Database Diagrams
Tables
aspnet_Applicatio
aspnet_Membersh
aspnet_Paths
aspnet_Personaliz
aspnet_Personaliz
aspnet_Profile
aspnet_Roles
aspnet_SchemaVe
aspnet_Users
aspnet_UsersInRo
aspnet_WebEvent
Views
Stored Procedures
Functions
Synonyms
Types
Toolbox
Server Explorer
aspnet_Member...ASPNETDB.MDF)
1 of 3
Solution Explorer - C:\...\Me...
Solution 'Membership' (1 project)
C:\...\Membership\
App_Data
CreateUser.aspx
web.config
Error List
0 Errors
0 Warnings
0 Messages
Description
File
Line
Column
Project
Error List
Output
Properties
CSS Properties
Ready

	ApplicationId	UserId	Password	PasswordFormat	PasswordSalt	Mobi
▶	a477-dc2e94f6097c	34decbb6-90a6-...	hnXc0vM+Z2ML...	1	NY/gDc0DgfGo...	NULL
	86081115-4bb0-...	180de2ce-42c0-...	agx+yabX96ypn...	1	I/1wyc/0NkbXw...	NULL
	86081115-4bb0-...	b68e277c-c6fe-4...	WcaqPN4Beq/e...	1	NPk+hDk9Qo7...	NULL
✳	NULL	NULL	NULL	NULL	NULL	NULL

[Qry] Query

(Name)	Query
Database Nan	C:\USERS\BILL\DOC
Destination Ta	
Distinct Value	No
GROUP BY Ext	<None>
Output All Co	Yes
Query Parame	No parameters have
Server Name	lipper-stl-lap\f5e22
SQL Comment	

(Name)

Figure 16-4

A note regarding the passwords used in ASP.NET 3.5: If you are having difficulty entering users because of a password error, it might be because ASP.NET requires strong passwords by default. All passwords input into the system must be at least seven characters, and contain at least one non-alphanumeric character (such as `[ ]`, `!`, `@`, `#`, or `$`). Whew! An example password of this combination is:

```
Bevjen7777$
```

Although this type of password is a heck of a lot more secure, a password like this is sometimes difficult to remember. You can actually change the behavior of the membership provider so that it doesn't require such difficult passwords by reworking the membership provider in the `web.config` file, as illustrated in Listing 16-4.

Listing 16-4: Modifying the membership provider in web.config

```
<configuration>
   <system.web>
```

```
        <membership>
           <providers>
              <clear />
              <add name="AspNetSqlMembershipProvider"
               type="System.Web.Security.SqlMembershipProvider, System.Web,
                 Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
               connectionStringName="LocalSqlServer"
               requiresQuestionAndAnswer="false"
               requiresUniqueEmail="true"
               passwordFormat="Hashed"
               minRequiredNonalphanumericCharacters="0"
               minRequiredPasswordLength="3" />
           </providers>
        </membership>

     </system.web>
  </configuration>
```

In this example, you have reworked the membership provider for SQL Server so that it does not actually require any non-alphanumeric characters and allows passwords as small as three characters in length. You do this by using the `minRequiredNonalphanumericCharacters` and `minRequiredPasswordLength` attributes. With these in place, you can now create users with these password rules as set forth in these configuration settings. Modifying the membership provider is covered in more detail later in this chapter.

Working with the CreateUserWizard Control

When you work with the CreateUserWizard control, be aware of the `ContinueButtonClick()` and the `CreatedUser()` events. The `ContinueButtonClick()` event is triggered when the Continue button on the second page is clicked after the user has been successfully created (see Listing 16-5).

Listing 16-5: The ContinueButtonClick event

VB

```
Protected Sub CreateUserWizard1_ContinueButtonClick(ByVal sender As Object, _
   ByVal e As System.EventArgs)

      Response.Redirect("Default.aspx")
End Sub
```

C#

```
protected void CreateUserWizard1_ContinueButtonClick(object sender, EventArgs e)
{
   Response.Redirect("Default.aspx");
}
```

In this example, after the user has been added to the membership service through the form provided by the CreateUserWizard control, she can click the Continue button to be redirected to another page in the application. This is done with a simple `Response.Redirect` statement. Remember when you use this event, that you must add an `OnContinueButtonClick = "CreateUserWizard1_ContinueButtonClick"` to the `<asp:CreateUserWizard>` control.

The `CreatingUser()` event is triggered when a user is successfully created in the data store. The use of this event is shown in Listing 16-6.

Listing 16-6: The CreateUser event

VB

```
Protected Sub CreateUserWizard1_CreatingUser(ByVal sender As Object, _
   ByVal e As System.EventArgs)

      ' Code here
End Sub
```

C#

```
protected void CreateUserWizard1_CreatingUser(object sender, EventArgs e)
{
   // Code here
}
```

Use this event if you want to take any additional actions when a user is registered to the service.

Incorporating Personalization Properties in the Registration Process

As you saw in the previous chapter on personalization, it is fairly simple to use the personalization management system that comes with ASP.NET 3.5 and store user-specific details. The registration process provided by the CreateUserWizard control is an ideal spot to retrieve this information from the user to store directly in the personalization system. The retrieval is not too difficult to incorporate into your code.

The first step, as you learned in the previous chapter on personalization, is to have some personalization points defined in the application's `web.config` file. This is shown in Listing 16-7.

Listing 16-7: Creating personalization properties in the web.config file

```
<configuration>
  <system.web>

     <profile>

        <properties>

           <add name="FirstName" />
           <add name="LastName" />
           <add name="LastVisited" />
           <add name="Age" />
           <add name="Member" />

        </properties>

      </profile>

  </system.web>
</configuration>
```

Now that these properties are defined in the `web.config` file, you can use them when you create users in the ASP.NET membership system. Again, using the CreateUserWizard control, you can create a process that requires the user to enter his preferred username and password in the first step, and then the second step asks for these custom-defined personalization points. Listing 16-8 shows a CreateUserWizard control that incorporates this idea.

Listing 16-8: Using personalization properties with the CreateUserWizard control

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub CreateUserWizard1_CreatedUser(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Dim pc As ProfileCommon = New ProfileCommon()
        pc.Initialize(CreateUserWizard1.UserName.ToString(), True)

        pc.FirstName = Firstname.Text
        pc.LastName = Lastname.Text
        pc.Age = Age.Text

        pc.Save()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Creating Users with Personalization</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:CreateUserWizard ID="CreateUserWizard1" Runat="server"
         BorderWidth="1px" BorderColor="#FFDFAD" BorderStyle="Solid"
         BackColor="#FFFBD6" Font-Names="Verdana"
         LoginCreatedUser="true" OnCreatedUser="CreateUserWizard1_CreatedUser" >
            <WizardSteps>
                <asp:WizardStep ID="WizardStep1" Runat="server"
                 Title="Additional Information" StepType="Start">
                    <table width="100%"><tr><td>
                    Firstname: </td><td>
                    <asp:TextBox ID="Firstname" Runat="server"></asp:TextBox>
                    </td></tr><tr><td>
                    Lastname: </td><td>
                    <asp:TextBox ID="Lastname" Runat="server"></asp:TextBox>
                    </td></tr><tr><td>
                    Age: </td><td>
                    <asp:TextBox ID="Age" Runat="server"></asp:TextBox>
                    </td></tr></table>
                </asp:WizardStep>
                <asp:CreateUserWizardStep Runat="server"
                 Title="Sign Up for Your New Account">
                </asp:CreateUserWizardStep>
                <asp:CompleteWizardStep Runat="server" Title="Complete">
                </asp:CompleteWizardStep>
            </WizardSteps>
            <StepStyle BorderColor="#FFDFAD" Font-Names="Verdana"
             BackColor="#FFFBD6" BorderStyle="Solid"
             BorderWidth="1px"></StepStyle>
            <TitleTextStyle Font-Bold="True" BackColor="#990000"
```

Continued

```
            ForeColor="White"></TitleTextStyle>
        </asp:CreateUserWizard>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void CreateUserWizard1_CreatedUser(object sender, EventArgs e)
    {
        ProfileCommon pc = new ProfileCommon();
        pc.Initialize(CreateUserWizard1.UserName.ToString(), true);

        pc.FirstName = Firstname.Text;
        pc.LastName = Lastname.Text;
        pc.Age = Age.Text;

        pc.Save();
    }
</script>
```

With this change to the standard registration process as is defined by a default instance of the CreateUserWizard control, your registration system now includes the request for properties stored and retrieved using the `ProfileCommon` object. Then, using the `ProfileCommon.Initialize()` method, you initialize the property values for the current user. Next, you set the property values using the strongly typed access to the profile properties available via the `ProfileCommon` object. After all the values have been set, you use the `Save()` method to finalize the process.

You can define a custom step within the CreateUserWizard control by using the `<WizardSteps>`element. Within this element, you can construct a series of registration steps in whatever fashion you choose. From the `<WizardSteps>`section, shown in Listing 16-8, you can see that three steps are defined. The first is the custom step in which the end user's personalization properties are requested with the `<asp:WizardStep>` control. Within the `<asp:WizardStep>`control, a table is laid out and a custom form is created.

Two additional steps are defined within Listing 16-7: a step to create the user (using the `<asp:CreateUserWizardStep>` control) and a step to confirm the creation of a new user (using the `<asp:CompleteWizardStep>` control). The order in which these steps appear is the order in which they are presented to the end user.

After the steps are created the way you want, you can then store the custom properties using the CreateUserWizard control's `CreatedUser()` event:

```
Protected Sub CreateUserWizard1_CreatedUser(ByVal sender As Object, _
   ByVal e As System.EventArgs)

    Dim pc As ProfileCommon = New ProfileCommon()
    pc.Initialize(CreateUserWizard1.UserName.ToString(), True)

    pc.FirstName = Firstname.Text
    pc.LastName = Lastname.Text
```

```
        pc.Age = Age.Text

        pc.Save()
    End Sub
```

You are not limited to having a separate step in which you ask for personal bits of information; you can incorporate these items directly into the `<asp:CreateUserWizardStep>` step itself. An easy way to do this is to switch to the Design view of your page and pull up the smart tag for the CreateUserWizard control. Then click the Customize Create User Step link (shown in Figure 16-5).

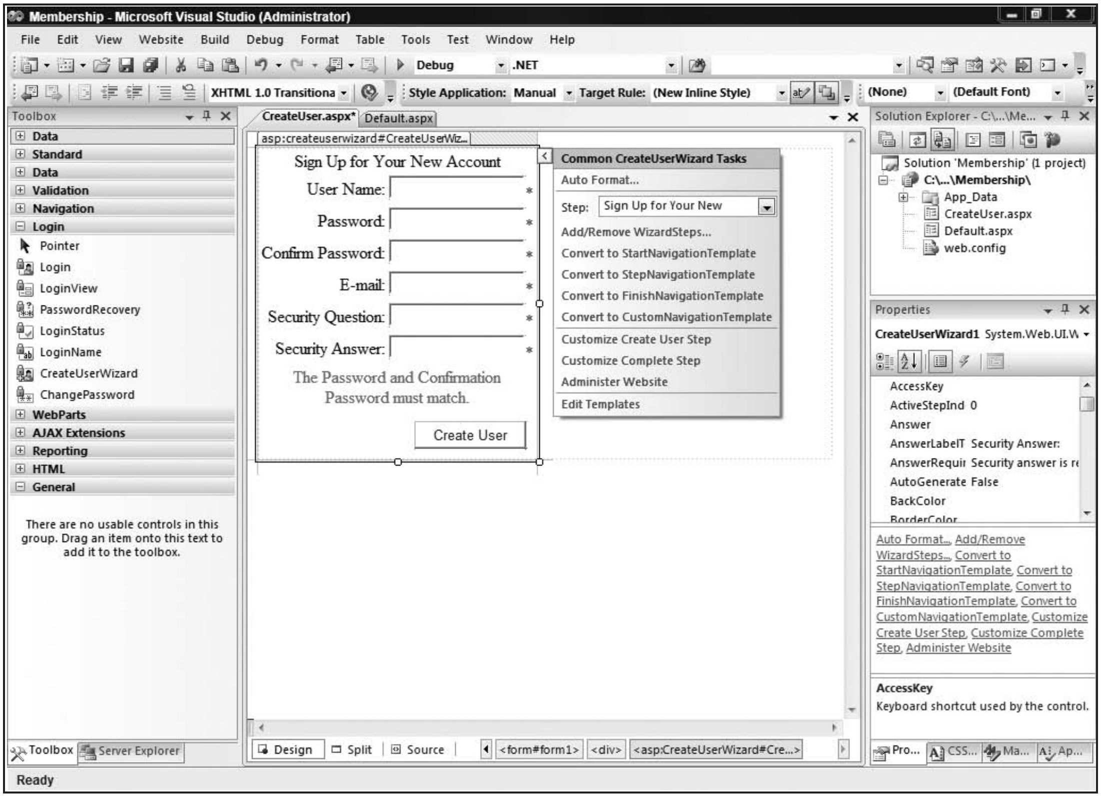
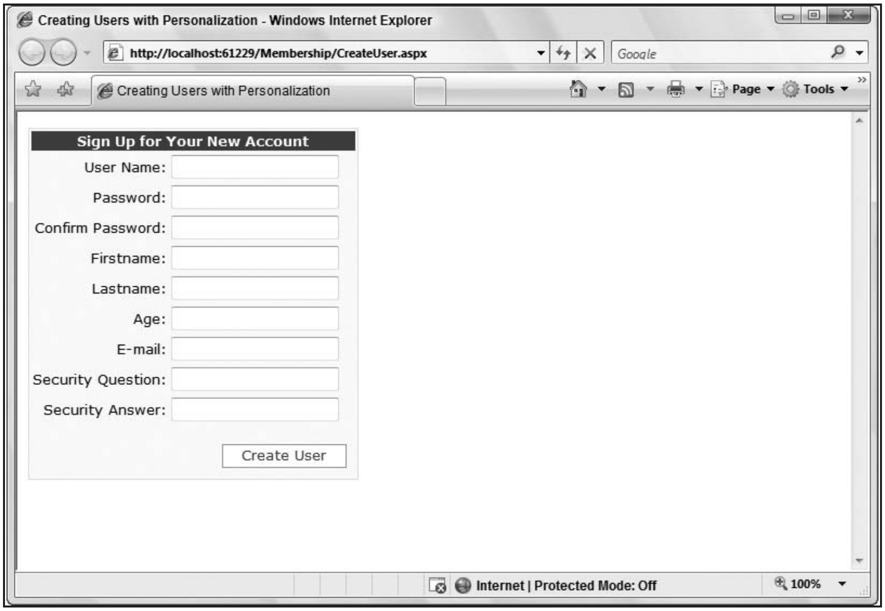

Membership - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Format Table Tools Test Window Help
Debug
.NET
XHTML 1.0 Transitiona
Style Application: Manual
Target Rule: (New Inline Style)
(None)
(Default Font)
Toolbox
Data
Standard
Data
Validation
Navigation
Login
Pointer
Login
LoginView
PasswordRecovery
LoginStatus
LoginName
CreateUserWizard
ChangePassword
WebParts
AJAX Extensions
Reporting
HTML
General
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
Toolbox
Server Explorer
CreateUser.aspx*
Default.aspx
asp:createuserwizard#CreateUserWiz...
Sign Up for Your New Account
User Name:
Password:
Confirm Password:
E-mail:
Security Question:
Security Answer:
The Password and Confirmation Password must match.
Create User
Common CreateUserWizard Tasks
Auto Format...
Step: Sign Up for Your New
Add/Remove WizardSteps...
Convert to StartNavigationTemplate
Convert to StepNavigationTemplate
Convert to FinishNavigationTemplate
Convert to CustomNavigationTemplate
Customize Create User Step
Customize Complete Step
Administer Website
Edit Templates
Design
Split
Source
<form#form1>
<div>
<asp:CreateUserWizard#Cre...>
Solution Explorer - C:\...\Me...
Solution 'Membership' (1 project)
C:\...\Membership\
App_Data
CreateUser.aspx
Default.aspx
web.config
Properties
CreateUserWizard1 System.Web.UI.W
AccessKey
ActiveStepInd 0
Answer
AnswerLabelT Security Answer:
AnswerRequir Security answer is r
AutoGenerate False
BackColor
BorderColor
Auto Format..., Add/Remove WizardSteps..., Convert to StartNavigationTemplate, Convert to StepNavigationTemplate, Convert to FinishNavigationTemplate, Convert to CustomNavigationTemplate, Customize Create User Step, Customize Complete Step, Administer Website
AccessKey
Keyboard shortcut used by the control.
Pro...
CSS...
Ma...
Ap...
Ready

Figure 16-5

Clicking on the Customize Create User Step details the contents of this particular step within a new `<ContentTemplate>` section that is now contained within the `<asp:CreateUserWizardStep>` control. Within the `<ContentTemplate>` element, you can now see the complete default form used for creating a new user. At this point, you are free to change the form by adding your own sections that request the end user's personal information. From this detailed form, you can also remove items. For instance, if you are not interested in asking for the security question and answer, you can remove these two items from the form (remember that you must disable the question-and-answer requirement in the membership provider definition). By changing this default form, you can completely customize the registration process for your end users (see Figure 16-6).


Creating Users with Personalization - Windows Internet Explorer
http://localhost:61229/Membership/CreateUser.aspx
Google
Creating Users with Personalization
Page
Tools
Internet | Protected Mode: Off
100%

Sign Up for Your New Account

User Name:

Password:

Confirm Password:

Firstname:

Lastname:

Age:

E-mail:

Security Question:

Security Answer:

Create User

Figure 16-6

Adding Users Programmatically

You are not limited to using only server controls to register or add new users to the membership service. ASP.NET 3.5 provides a Membership API for performing this task programmatically. This is ideal to create your own mechanics for adding users to the service — or if you are modifying a Web application that was created using ASP.NET 1.0/1.1.

The Membership API includes the `CreateUser()` method for adding users to the service. The `CreateUser()` method includes four possible signatures:

```
Membership.CreateUser(username As String, password As String)

Membership.CreateUser(username As String, password As String,
   email As String)

Membership.CreateUser(username As String, password As String,
   email As String, passwordQuestion As String,
   passwordAnswer As String, isApproved As Boolean,
   ByRef status As System.Web.Security.MembershipCreateStatus)

Membership.CreateUser(username As String, password As String,
   email As String, passwordQuestion As String,
   passwordAnswer As String, isApproved As Boolean, providerUserKey As Object
   ByRef status As System.Web.Security.MembershipCreateStatus)
```

You can use this method to create users. The nice thing about this method is that you are not required to create an instance of the `Membership` class; you use it directly. A simple use of the `CreateUser()` method is illustrated in Listing 16-9.

Listing 16-9: Creating users programmatically

VB

```
<%@ Page Language="VB" %>

<script runat="server">
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
   Try
      Membership.CreateUser(TextBox1.Text, TextBox2.Text)
      Label1.Text = "Successfully created user " & TextBox1.Text
   Catch ex As MembershipCreateUserException
      Label1.Text = "Error: " & ex.ToString()
   End Try
End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Creating a User</title>
</head>
<body>
    <form id="form1" runat="server">
        <h1>Create User</h1>
        <p>Username<br />
            <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
        </p>
        <p>Password<br />
            <asp:TextBox ID="TextBox2" Runat="server"
             TextMode="Password"></asp:TextBox>
        </p>
        <p>
            <asp:Button ID="Button1" Runat="server" Text="Create User"
             OnClick="Button1_Click" />
        </p>
        <p>
            <asp:Label ID="Label1" Runat="server"></asp:Label>
        </p>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        try
        {
            Membership.CreateUser(TextBox1.Text.ToString(),
               TextBox2.Text.ToString());
            Label1.Text = "Successfully created user " + TextBox1.Text;
        }
```

Continued

```
            catch (MembershipCreateUserException ex)
            {
                Label1.Text = "Error: " + ex.ToString();
            }
        }
    </script>
```

So, use either the CreateUserWizard control or the `CreateUser()` method found in the Membership API to create users for your Web applications with relative ease. This functionality was possible in the past with ASP.NET 1.0/1.1, but it was a labor-intensive task. With ASP.NET 2.0 or 3.5, you can create users either with a single control or with a single line of code.

From this bit of code, you can see that if a problem occurs when creating the user with the `CreateUser()` method, a `MembershipCreateUserException` is thrown. In this example, the exception is written to the screen within a Label server control. An example of an exception written to the screen is presented here:

```
Error: System.Web.Security.MembershipCreateUserException: The password-answer
supplied is invalid. at System.Web.Security.Membership.CreateUser(String username,
String password, String email) at System.Web.Security.Membership.CreateUser(String
username, String password) at ASP.default_aspx.Button1_Click(Object sender,
EventArgs e) in c:\Documents and Settings\BillEvjen\My Documents\Visual Studio
2008\WebSites\Membership\Default.aspx:line 10
```

You might not want such details sent to the end user. You might prefer to return a simpler message to the end user with something like the following construct:

```
Label1.Text = "Error: " & ex.Message.ToString();
```

This gives you results as simple as the following:

```
Error: The password-answer supplied is invalid.
```

You can also capture the specific error using the `MembershipCreateUserException` and returning something that might be a little more appropriate. An example of this is presented in Listing 16-10.

Listing 16-10: Capturing the specific MembershipCreateUserException value

VB

```
<%@ Page Language="VB" %>

<script runat="server">

    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        Try
            Membership.CreateUser(TextBox1.Text, TextBox2.Text)
            Label1.Text = "Successfully created user " & TextBox1.Text
        Catch ex As MembershipCreateUserException
            Select Case ex.StatusCode
                Case MembershipCreateStatus.DuplicateEmail
                    Label1.Text = "You have supplied a duplicate email address."
                Case MembershipCreateStatus.DuplicateUserName
                    Label1.Text = "You have supplied a duplicate username."
```

Continued

```
                Case MembershipCreateStatus.InvalidEmail
                    Label1.Text = "You have not supplied a proper email address."
                Case Else
                    Label1.Text = "ERROR: " & ex.Message.ToString()
            End Select
        End Try
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Creating a User</title>
</head>
<body>
    <form id="form1" runat="server">
        <h1>Create User</h1>
        <p>Username<br />
            <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
        </p>
        <p>Password<br />
            <asp:TextBox ID="TextBox2" Runat="server"
             TextMode="Password"></asp:TextBox>
        </p>
        <p>
            <asp:Button ID="Button1" Runat="server" Text="Create User"
             OnClick="Button1_Click" />
        </p>
        <p>
            <asp:Label ID="Label1" Runat="server"></asp:Label>
        </p>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        try
        {
            Membership.CreateUser(TextBox1.Text, TextBox2.Text);
            Label1.Text = "Successfully created user " + TextBox1.Text;
        }
        catch (MembershipCreateUserException ex)
        {
            switch(ex.StatusCode)
            {
                case MembershipCreateStatus.DuplicateEmail:
                    Label1.Text = "You have supplied a duplicate email address.";
                    break;
                case MembershipCreateStatus.DuplicateUserName:
                    Label1.Text = "You have supplied a duplicate username.";
```

Continued

```
                    break;
                case MembershipCreateStatus.InvalidEmail:
                    Label1.Text = "You have not supplied a proper email address.";
                    break;
                default:
                    Label1.Text = "ERROR: " + ex.Message.ToString();
                    break;
            }
        }
    }
</script>
```

In this case, you are able to look for the specific error that occurred in the `CreateUser` process. Here, this code is looking for only three specific items, but the list of available error codes includes the following:

- ☐ `MembershipCreateStatus.DuplicateEmail`
- ☐ `MembershipCreateStatus.DuplicateProviderUserKey`
- ☐ `MembershipCreateStatus.DuplicateUserName`
- ☐ `MembershipCreateStatus.InvalidAnswer`
- ☐ `MembershipCreateStatus.InvalidEmail`
- ☐ `MembershipCreateStatus.InvalidPassword`
- ☐ `MembershipCreateStatus.InvalidProviderUserKey`
- ☐ `MembershipCreateStatus.InvalidQuestion`
- ☐ `MembershipCreateStatus.InvalidUserName`
- ☐ `MembershipCreateStatus.ProviderError`
- ☐ `MembershipCreateStatus.Success`
- ☐ `MembershipCreateStatus.UserRejected`

In addition to giving better error reports to your users by defining what is going on, you can use these events to take any actions that might be required.

Changing How Users Register with Your Application

You determine how users register with your applications and what is required of them by the membership provider you choose. You will find a default membership provider and its applied settings are established within the `machine.config` file. If you dig down in the `machine.config` file on your server, you find the code shown in Listing 16-11.

Listing 16-11: Membership provider settings in the machine.config file

```
<membership>
   <providers>
      <add name="AspNetSqlMembershipProvider"
       type="System.Web.Security.SqlMembershipProvider, System.Web,
          Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
       connectionStringName="LocalSqlServer"
```

```
          enablePasswordRetrieval="false"
          enablePasswordReset="true"
          requiresQuestionAndAnswer="true"
          applicationName="/"
          requiresUniqueEmail="false"
          passwordFormat="Hashed"
          maxInvalidPasswordAttempts="5"
          minRequiredPasswordLength="7"
          minRequiredNonalphanumericCharacters="1"
          passwordAttemptWindow="10"
          passwordStrengthRegularExpression="" />
      </providers>
   </membership>
```

This section of the `machine.config` file shows the default membership provider that comes with ASP.NET 3.5 — named `AspNetSqlProvider`. If you are adding membership providers for server-wide use, add them to this `<membership>` section of the `machine.config` file; if you intend to use them for only a specific application instance, you can add them to your application's `web.config` file.

The important attributes of the SqlMembershipProvider definition include the `enablePasswordRetrieval`, `enablePasswordReset`, `requiresQuestionAndAnswer`, `requiresUniqueEmail`, and `PasswordFormat` attributes. The following table defines these attributes.

Attribute	**Description**
`enablePasswordRetrieval`	Defines whether the provider supports password retrievals. This attribute takes a `Boolean` value. The default value is `False`. When it is set to `False`, passwords cannot be retrieved although they can be changed with a new random password.
`enablePasswordReset`	Defines whether the provider supports password resets. This attribute takes a `Boolean` value. The default value is `True`.
`requiresQuestionAndAnswer`	Specifies whether the provider should require a question-and-answer combination when a user is created. This attribute takes a `Boolean` value, and the default value is `False`.
`requiresUniqueEmail`	Defines whether the provider should require a unique e-mail to be specified when the user is created. This attribute takes a `Boolean` value, and the default value is `False`. When set to `True`, only unique e-mail addresses can be entered into the data store.
`passwordFormat`	Defines the format in which the password is stored in the data store. The possible values include `Hashed`, `Clear`, and `Encrypted`. The default value is `Hashed`. Hashed passwords use SHA1, whereas encrypted passwords use Triple-DES encryption.

In addition to having these items defined in the `machine.config` file, you can also redefine them again (thus overriding the settings in the `machine.config`) in the `web.config` file.

Asking for Credentials

After you have users that can access your Web application using the membership service provided by ASP.NET, you can then give these users the means to log in to the site. This requires little work on your part. Before you learn about the controls that enable users to access your applications, you should make a few more modifications to the `web.config` file.

Turning Off Access with the <authorization> Element

After you make the changes to the `web.config` file by adding the `<authorization>` and `<forms>` elements (Listings 16-1 and 16-2), your Web application is accessible to each and every user that browses to any page your application contains. To prevent open access, you have to deny unauthenticated users access to the pages of your site.

Denying unauthenticated users access to your site is illustrated in Listing 16-12.

Listing 16-12: Denying unauthenticated users

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <system.web>
        <authentication mode="Forms" />
        <authorization>
           <deny users="?" />
        </authorization>
    </system.web>
</configuration>
```

Using the `<authorization>` and `<deny>` elements, you can deny specific users access to your Web application — or (as in this case) simply deny every unauthenticated user (this is what the question mark signifies).

Now that everyone but authenticated users has been denied access to the site, you want to make it easy for viewers of your application to become authenticated users. To do so, use the Login server control.

Using the Login Server Control

The Login server control enables you to turn unauthenticated users into authenticated users by allowing them to provide login credentials that can be verified in a data store of some kind. In the examples so far, you have used Microsoft SQL Server Express Edition as the data store, but you can just as easily use the full-blown version of Microsoft's SQL Server (such as Microsoft's SQL Server 7.0, 2000, 2005, or 2008).

The first step in using the Login control is to create a new Web page titled `Login.aspx`. This is the default page to which unauthenticated users are redirected to obtain their credentials. Remember that you can change this behavior by changing the value of the `<forms>` element's `loginUrl` attribute in the `web.config` file.

The `Login.aspx` page simply needs an `<asp:Login>` control to give the end user everything he needs to become authenticated, as illustrated in Listing 16-13.

Listing 16-13: Providing a login for the end user using the Login control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Login Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Login ID="Login1" Runat="server">
        </asp:Login>
    </form>
</body>
</html>
```

In the situation established here, if the unauthenticated user hits a different page in the application, he is redirected to the `Login.aspx` page. You can see how ASP.NET tracks the location in the URL from the address bar in the browser:

```
http://localhost:18436/Membership/Login.aspx?ReturnUrl=%2fMembership%2fDefault.aspx
```

The login page, using the Login control, is shown in Figure 16-7.

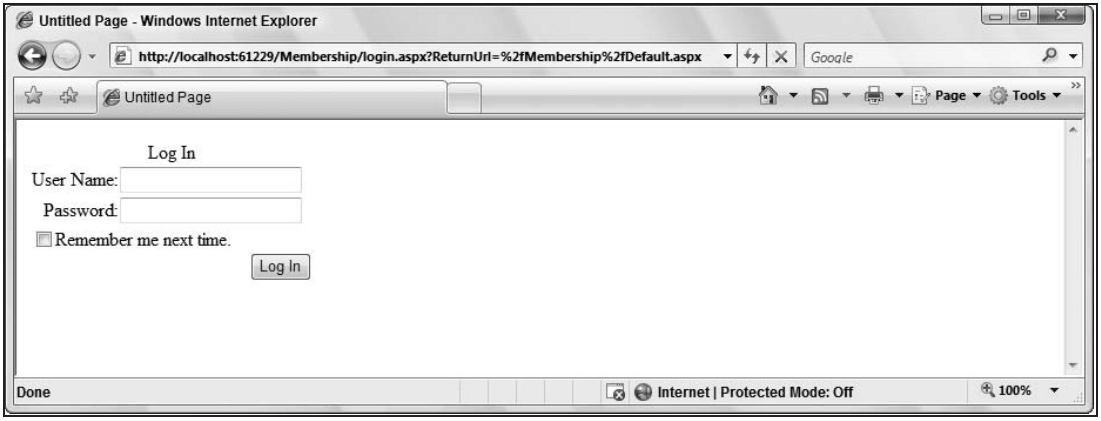

Untitled Page - Windows Internet Explorer
http://localhost:61229/Membership/login.aspx?ReturnUrl=%2fMembership%2fDefault.aspx
Google
Untitled Page
Page
Tools
Log In
User Name:
Password:
☐ Remember me next time.
Log In
Done
Internet | Protected Mode: Off
100%

Figure 16-7

From this figure, you can see that the Login control asks the user for a username and password. A check box allows a cookie to be stored on the client machine. This cookie enables the end user to bypass future logins. You can remove the check box and related text created to remember the user by setting the Login control's `DisplayRememberMe` property to `False`.

In addition to the `DisplayRememberMe` property, you can work with this aspect of the Login control by using the `RememberMeText` and the `RememberMeSet` properties. The `RememberMeText` property is pretty self-explanatory because its value simply defines the text set next to the check box. The `RememberMeSet` property, however, is fairly interesting. The `RememberMeSet` property takes a `Boolean` value (by default, it is set to `False`) that specifies whether to set a persistent cookie on the client's machine after a user has logged in using the Login control. If set to `True` when the `DisplayRememberMe` property is also set to `True`, the check box is simply checked by default when the Login control is generated in the browser. If the `DisplayRememberMe` property is set to `False` (meaning the end user does not see the check box or cannot select the option of persisting the login cookie) and the `RememberMeSet` is set to `True`, a cookie is set on the user's machine automatically without the user's knowledge or choice in the matter. You should think carefully about taking this approach because end users sometimes use public computers, and this method would mean you are setting authorization cookies on public machines.

This cookie remains on the client's machine until the user logs out of the application (if this option is provided). With the persisted cookie, and assuming the end user has not logged out of the application, the user never needs to log in again when he returns to the application because his credentials are provided by the contents found in the cookie. After the end user has logged in to the application, he is returned to the page he originally intended to access.

You can also modify the look-and-feel of the Login control just as you can for the other controls. One way to do this is by clicking the Auto Format link in the control's smart tag. There you find a list of options for modifying the look-and-feel of the control (see Figure 16-8).

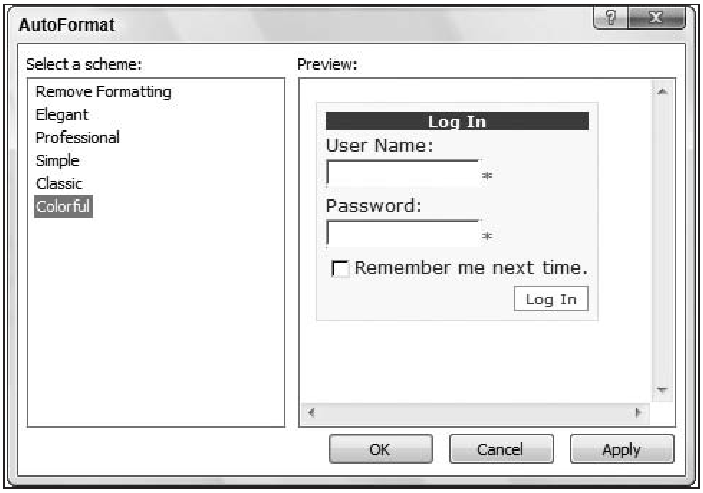


Figure 16-8

Select the Colorful option, for example, and the code is modified. Listing 16-14 shows the code generated for this selection.

Listing 16-14: A formatted Login control

```
<asp:Login ID="Login1" Runat="server" BackColor="#FFFBD6"
 BorderColor="#FFDFAD" BorderPadding="4" BorderStyle="Solid"
 BorderWidth="1px" Font-Names="Verdana" Font-Size="0.8em"
 ForeColor="#333333" TextLayout="TextOnTop">
```

```
        <TextBoxStyle Font-Size="0.8em" />
        <LoginButtonStyle BackColor="White" BorderColor="#CC9966"
         BorderStyle="Solid" BorderWidth="1px" Font-Names="Verdana"
         Font-Size="0.8em" ForeColor="#990000" />
        <InstructionTextStyle Font-Italic="True" ForeColor="Black" />
        <TitleTextStyle BackColor="#990000" Font-Bold="True" Font-Size="0.9em"
         ForeColor="White" />
  </asp:Login>
```

From this listing, you can see that there are a number of subelements that are used to modify particular items displayed by the control. The available styling elements for the Login control include the following:

- `<CheckboxStyle>`
- `<FailureTextStyle>`
- `<HyperLinkStyle>`
- `<InstructionTextStyle>`
- `<LabelStyle>`
- `<LoginButtonStyle>`
- `<TextBoxStyle>`
- `<TitleTextStyle>`
- `<ValidatorTextStyle>`

The Login control has numerous properties that allow you to alter how the control appears and behaves. An interesting change you can make is to add some links at the bottom of the control to provide access to additional resources. With these links, you can give users the capability to get help or register for the application so that they can be provided with any login credentials.

You can provide links to do the following:

- Redirect users to a help page using the `HelpPageText`, `HelpPageUrl`, and `HelpPageIconUrl` properties.
- Redirect users to a registration page using the `CreateUserText`, `CreateUserUrl`, and `CreateUserIconUrl` properties.
- Redirect users to a page that allows them to recover their forgotten passwords using the `PasswordRecoveryText`, `PasswordRecoveryUrl`, and `PasswordRecoveryIconUrl` properties.

When used, the Login control looks like what is shown in Figure 16-9.

Logging In Users Programmatically

Besides using the pre-built mechanics of the Login control, you can also perform this task programmatically using the Membership class. To validate credentials that you receive, you use the `ValidateUser()` method of this class. The `ValidateUser()` method takes a single signature:

```
Membership.ValidateUser(username As String, password As String)
```

This method is illustrated in Listing 16-15.

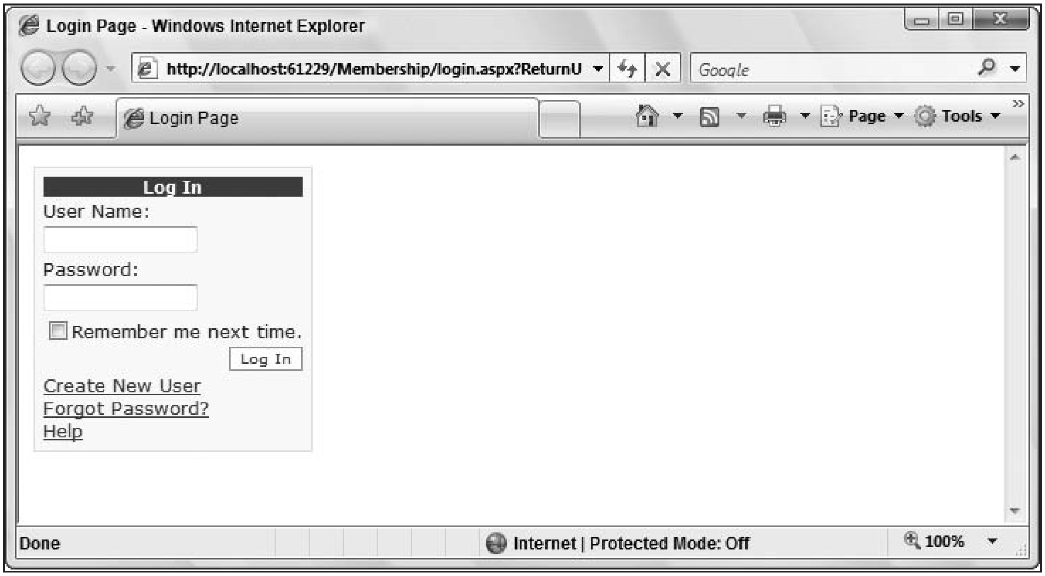

Login Page - Windows Internet Explorer
http://localhost:61229/Membership/login.aspx?ReturnU
Google
Login Page
Page
Tools
Log In
User Name:
Password:
☐ Remember me next time.
Log In
Create New User
Forgot Password?
Help
Done
Internet | Protected Mode: Off
100%

Figure 16-9

Listing 16-15: Validating a user's credentials programmatically

VB

```
If Membership.ValidateUser(TextBox1.Text, TextBox2.Text) Then
   FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, False)
Else
   Label1.Text = "You are not registered with the site."
End If
```

C#

```
if (Membership.ValidateUser(TextBox1.Text, TextBox2.Text) {
   FormsAuthentication.RedirectFromLoginPage(TextBox1.Text.ToString(), false);
}
else {
   Label1.Text = "You are not registered with the site.";
}
```

The `ValidateUser()` method returns a `Boolean` value of `True` if the user credentials pass the test and `False` if they do not. From the code snippet in Listing 16-15, you can see that end users whose credentials are verified as correct are redirected from the login page using the `RedirectFromLoginPage()` method. This method takes the username and a `Boolean` value that specifies whether the credentials are persisted through a cookie setting.

Locking Out Users Who Provide Bad Passwords

When providing a user login form in any application you build, always guard against repeated bogus password attempts. If you have a malicious end user who knows a username, he may try to access the application by repeatedly trying different passwords. You want to guard against this kind of activity. You don't want to allow this person to try hundreds of possible passwords with this username.

ASP.NET has built-in protection against this type of activity. If you look in the aspnet_Membership table, you see two columns focused on protecting against this. These columns are `FailedPasswordAttemptCount` and `FailedPasswordAttemptWindowStart`.

By default, a username can be used with an incorrect password in a login attempt only five times within a 10-minute window. On the fifth failed attempt, the account is locked down. This is done in ASP.NET by setting the `IsLockedOut` column to `True`.

You can actually control the number of password attempts that are allowed and the length of the attempt window for your application. These two items are defined in the SqlMembershipProvider declaration in the `machine.config` file. You can change the values either in the server-wide configuration files or in your application's `web.config` file. Changing these values in your `web.config` file is presented in Listing 16-16.

Listing 16-16: Changing the values for password attempts in the provider declaration

```
<configuration>
   <system.web>

      <membership defaultProvider="AspNetSqlMembershipProvider">
        <providers>
          <clear />
          <add connectionStringName="LocalSqlServer"
           applicationName="/"
           maxInvalidPasswordAttempts="3"
           passwordAttemptWindow="15"
           name="AspNetSqlMembershipProvider"
           type="System.Web.Security.SqlMembershipProvider, System.Web,
             Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
        </providers>
      </membership>

   </system.web>
</configuration>
```

To determine the number of password attempts that are allowed, use `maxInvalidPasswordAttempts`. This example changes the value to `3`, meaning that users are allowed to enter an incorrect password three times before being locked out (within the time window defined). The default value of the `maxInvalidPasswordAttempts` attribute is `5`. You can set the time allowed for bad password attempts to `15` minutes using the `passwordAttemptWindow` attribute. The default value of this attribute is `10`, so an extra five minutes is added.

Now that these items are in place, the next step is to test it. Listing 16-17 provides you with an example of the test. It assumes you have an application established with a user already in place.

Listing 16-17: A sample page to test password attempts

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)
```

Continued

```
        If CheckBox1.Checked = True Then
            Dim user As MembershipUser = Membership.GetUser(TextBox1.Text)
            user.UnlockUser()
        End If

        If Membership.ValidateUser(TextBox1.Text, TextBox2.Text) Then
            Label1.Text = "You are logged on!"
        Else
            Dim user As MembershipUser = Membership.GetUser(TextBox1.Text)
            Label1.Text = "Locked out value: " & user.IsLockedOut.ToString()
        End If
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Login Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <h1>Login User</h1>
        <p>
            <asp:CheckBox ID="CheckBox1" runat="server" Text="Unlock User" />
        </p>
        <p>
            Username<br />
            <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
        </p>
        <p>Password<br />
            <asp:TextBox ID="TextBox2" Runat="server"
             TextMode="Password"></asp:TextBox>
        </p>
        <p>
            <asp:Button ID="Button1" Runat="server" Text="Login"
             OnClick="Button1_Click" />
        </p>
        <p>
            <asp:Label ID="Label1" Runat="server"></asp:Label>
        </p>
    </div>
    </form>
</body>
</html>
```

C#

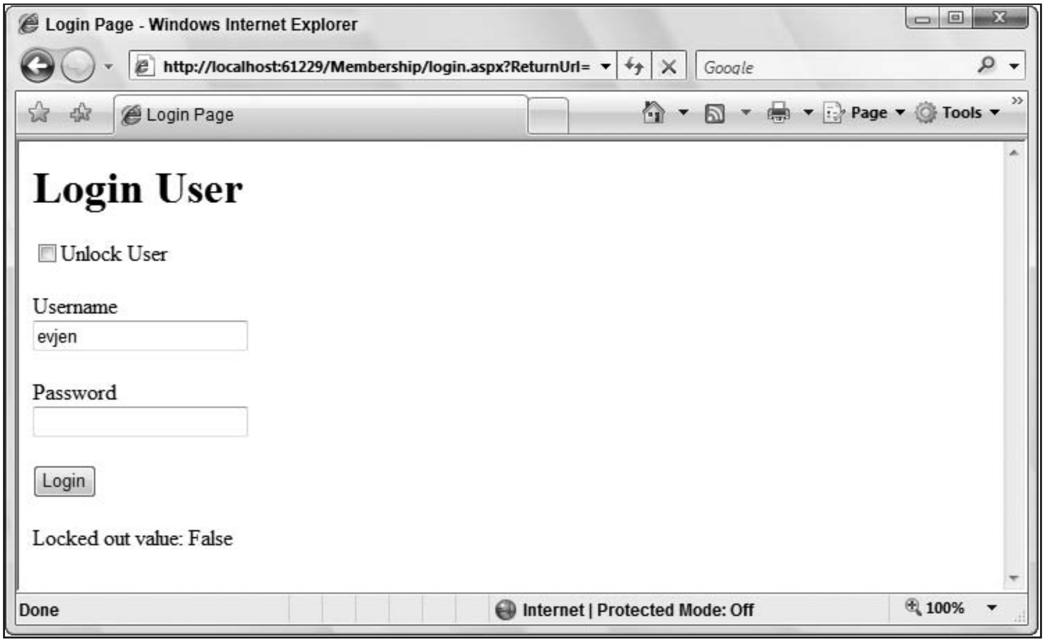
```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        if (CheckBox1.Checked == true)
```

Continued

```
        {
            MembershipUser user = Membership.GetUser(TextBox1.Text);
            user.UnlockUser();
        }

        if (Membership.ValidateUser(TextBox1.Text, TextBox2.Text))
        {
            Label1.Text = "You are logged on!";
        }
        else
        {
            MembershipUser user = Membership.GetUser(TextBox1.Text);
            Label1.Text = "Locked out value: " + user.IsLockedOut.ToString();
        }
    }
</script>
```

This page contains two text boxes: one for the username and another for the password. Above these, however, is a check box that can be used to unlock a user after you have locked down the account because of bad password attempts.

If you run this page and enter three consecutive bad passwords for your user, you get the results presented in Figure 16-10.


Login Page - Windows Internet Explorer
http://localhost:61229/Membership/login.aspx?ReturnUrl=
Google
Login Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

Login User

☐ Unlock User

Username
evjen

Password

Login

Locked out value: False

Figure 16-10

The `IsLockedOut` property is read through an instantiation of the `MembershipUser` object. This object allows you programmatic access to the user data points contained in the aspnet_Membership table. In this case, the `IsLockedOut` property is retrieved and displayed to the screen. The `MembershipUser` object also exposes a lot of available methods — one of which is the `UnlockUser()` method. This method is invoked if the check box is checked in the button-click event.

Working with Authenticated Users

After users are authenticated, ASP.NET 3.5 provides a number of different server controls and methods that you can use to work with the user details. Included in this collection of tools are the LoginStatus and the LoginName controls.

The LoginStatus Server Control

The LoginStatus server control enables users to click a link to log in or log out of a site. For a good example of this control, remove the `<deny>` element from the `web.config` file so that the pages of your site are accessible to unauthenticated users. Then code your `Default.aspx` page so that it is similar to the code shown in Listing 16-18.

Listing 16-18: Login and logout features of the LoginStatus control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Login or Logout</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:LoginStatus ID="LoginStatus1" Runat="server" />
    </form>
</body>
</html>
```

Running this gives you a simple page that contains only a hyperlink titled `Login`, as shown in Figure 16-11.

Clicking the `Login` hyperlink forwards you to the `Login.aspx` page where you provide your credentials. After the credentials are provided, you are redirected to the `Default.aspx` page — although now the page includes a hyperlink titled `Logout` (see Figure 16-12). The LinkStatus control displays one link when the user is unauthenticated and another link when the user is authenticated. Clicking the `Logout` hyperlink logs out the user and redraws the `Default.aspx` page — but with the `Login` hyperlink in place.

The LoginName Server Control

The LoginName server control enables you to display the username of the authenticated user. This is a common practice today. For an example of this, change the `Default.aspx` page so that it now includes the authenticated user's login name when that user is logged in, as illustrated in Listing 16-19.

Listing 16-19: Displaying the username of the authenticated user

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head runat="server">
    <title>Login or Logout</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:LoginStatus ID="LoginStatus1" Runat="server" />
        <p><asp:LoginName ID="LoginName1" Runat="server"
            Font-Bold="True" Font-Size="XX-Large" /></p>
    </form>
</body>
</html>
```

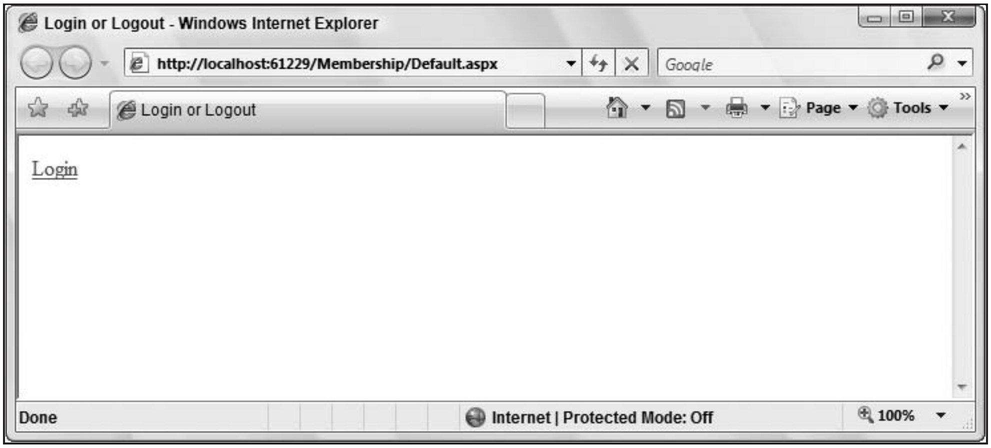

Login or Logout - Windows Internet Explorer
http://localhost:61229/Membership/Default.aspx
Google
Login or Logout
Page
Tools
Login
Done
Internet | Protected Mode: Off
100%

Figure 16-11

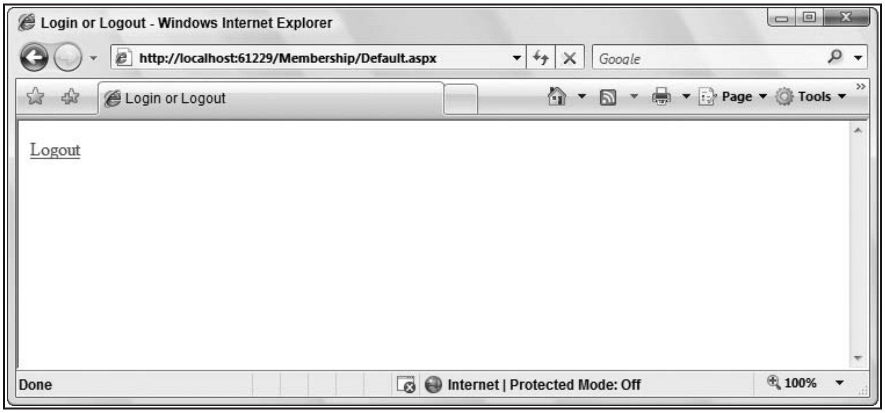

Login or Logout - Windows Internet Explorer
http://localhost:61229/Membership/Default.aspx
Google
Login or Logout
Page
Tools
Logout
Done
Internet | Protected Mode: Off
100%

Figure 16-12

When the user logs in to the application and is returned to the `Default.aspx` page, he sees his username displayed, as well as the hyperlink generated by the LoginStatus control (see Figure 16-13).

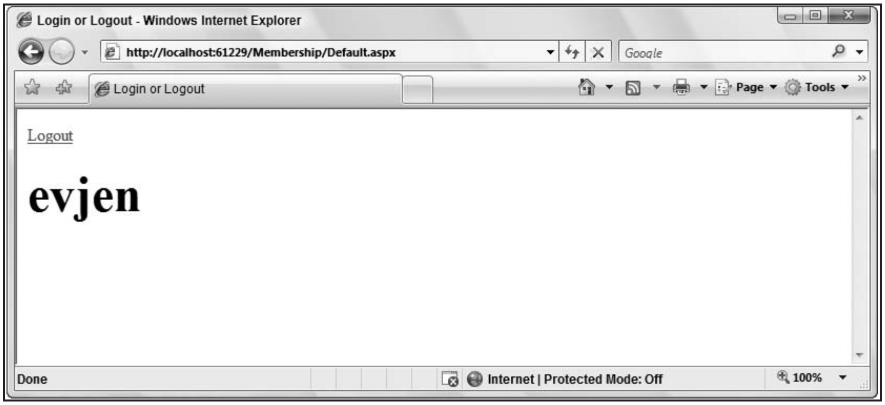

Login or Logout - Windows Internet Explorer
http://localhost:61229/Membership/Default.aspx
Google
Login or Logout
Page
Tools
Logout
evjen
Done
Internet | Protected Mode: Off
100%

Figure 16-13

In addition to just showing the username of the logged in user, you can also add text by using the LoginName control's `FormatString` property. For instance, to provide a welcome message along with the username, you construct the LoginName control as follows:

```
<asp:LoginName ID="LoginName1" Runat="Server"
 FormatString="Welcome to our Website {0}!" />
```

You can also simply use the following construction in one of the page events. (This is shown in VB; if you are using C#, add a semicolon at the end of the line.)

```
LoginName1.FormatString = "Welcome to the site {0}!"
```

When the page is generated, ASP.NET replaces the `{0}` part of the string with the username of the logged-in user. This provides you with a result similar to the following:

```
Welcome to the site evjen!
```

If you do not want to show the username when using the LoginName control, simply omit the `{0}` aspect of the string. The control then places the `FormatString` property's value on the page.

Showing the Number of Users Online

One cool feature of the membership service is that you can display how many users are online at a given moment. This is an especially popular option for a portal or a forum that wishes to impress visitors to the site with its popularity.

To show the number of users online, you use the `GetNumberOfUsersOnline` method provided by the `Membership` class. You can add to the `Default.aspx` page shown in Figure 16-10 with the code illustrated in Listing 16-20.

Listing 16-20: Displaying the number of users online

VB

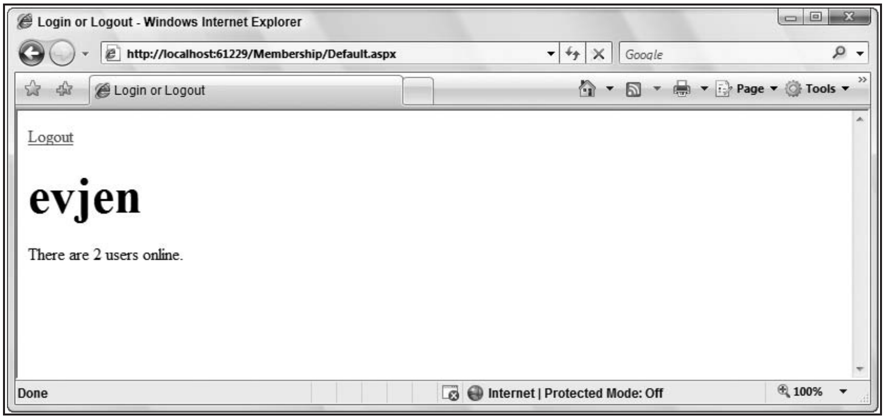
```
<%@ Page Language="VB" %>

<script runat="server">
```

```
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = Membership.GetNumberOfUsersOnline().ToString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Login or Logout</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:LoginStatus ID="LoginStatus1" Runat="server" />
        <p><asp:LoginName ID="LoginName1" Runat="server"
            Font-Bold="True" Font-Size="XX-Large" /></p>
        <p>There are <asp:Label ID="Label1" Runat="server" Text="0" />
           users online.</p>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
   protected void Page_Load(object sender, EventArgs e)
   {
      Label1.Text = Membership.GetNumberOfUsersOnline().ToString();
   }
</script>
```

When the page is generated, it displays the number of users who have logged on in the last 15 minutes. An example of what is generated is shown in Figure 16-14.


Login or Logout - Windows Internet Explorer
http://localhost:61229/Membership/Default.aspx
Google
Login or Logout
Page
Tools
Logout

evjen

There are 2 users online.

Done
Internet | Protected Mode: Off
100%

Figure 16-14

You can see that two users have logged on in the last 15 minutes. This 15-minute period is determined in the `machine.config` file from within the `<membership>` element:

```
<membership userIsOnlineTimeWindow="15" >
</membership>
```

By default, the `userIsOnlineTimeWindow` is set to `15`. The number is specified here in minutes. To increase the time window, you simply increase this number. In addition to specifying this number from within the `machine.config` file, you can also set this number in the `web.config` file.

Dealing with Passwords

Many of us seem to spend our lives online and have username/password combinations for many different Web sites on the Internet. For this reason, end users forget passwords or want to change them every so often. ASP.NET provides a couple of server controls that work with the membership service so that end users can either change their passwords or retrieve forgotten passwords.

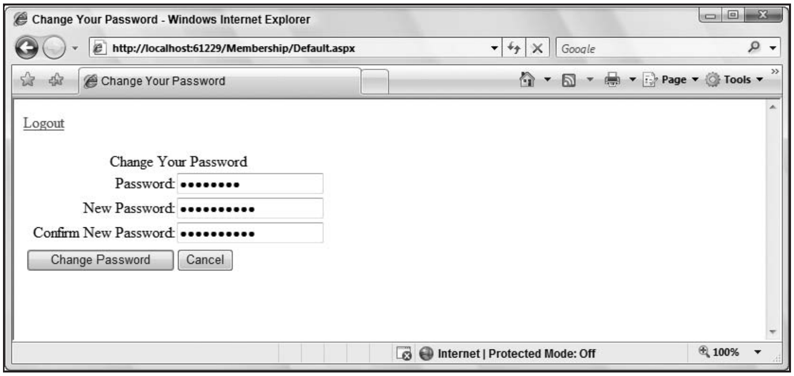
The ChangePassword Server Control

The ChangePassword server control enables end users to change their passwords directly in the browser. Listing 16-21 shows a use of the ChangePassword control.

Listing 16-21: Allowing users to change passwords

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Change Your Password</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:LoginStatus ID="LoginStatus1" Runat="server" />
        <p><asp:ChangePassword ID="ChangePassword1" Runat="server">
           </asp:ChangePassword><p>
    </form>
</body>
</html>
```

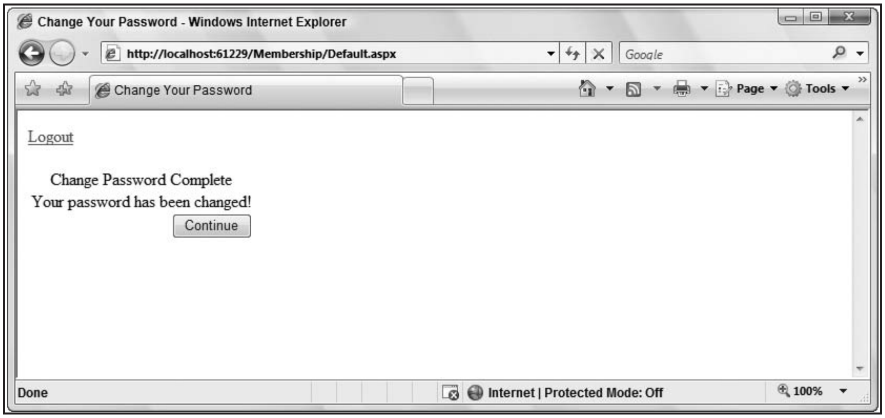
This is a rather simple use of the `<asp:ChangePassword>` control. Running this page produces the results shown in Figure 16-15.


Change Your Password - Windows Internet Explorer
http://localhost:61229/Membership/Default.aspx
Google
Change Your Password
Page
Tools
Logout
Change Password
Cancel
Internet | Protected Mode: Off
100%

Change Your Password
Password: ••••••••
New Password: •••••••••••
Confirm New Password: •••••••••••

Figure 16-15

The ChangePassword control produces a form that asks for the previous password. It also requires the end user to type the new password twice. Clicking the Change Password button launches an attempt to change the password if the user is logged in. If the end user is not logged in to the application yet, he or she is redirected to the login page. Only a logged-in user can change a password. After the password is changed, the end user is notified (see Figure 16-16).


Change Your Password - Windows Internet Explorer
http://localhost:61229/Membership/Default.aspx
Google
Change Your Password
Page
Tools
Logout
Continue
Done
Internet | Protected Mode: Off
100%

Change Password Complete
Your password has been changed!

Figure 16-16

Remember that end users are allowed to change their passwords because the `enablePasswordReset` attribute of the membership provider is set to `true`. To deny this capability, set the `enablePasswordReset` attribute to `false`.

You can also specify rules on how the passwords must be constructed when an end user attempts to change her password. For instance, you might want to require that the password contain more than a certain number of characters or that it use numbers and/or special characters in addition to alpha-characters. Using the `NewPasswordRegularExpression` attribute, you can specify the construction required for the new password, as shown here:

```
NewPasswordRegularExpression='@\"(?=.{6,})(?=(.*\d){1,})(?=(.*\W){1,})'
```

Any new passwords created by the end user are checked against this regular expression. If there isn't a match, you can use the `NewPasswordRegularExpressionErrorMessage` attribute (one of the lengthier names for an attribute in ASP.NET) to cause an error message to appear within the control output.

The PasswordRecovery Server Control

People simply forget their passwords. For this reason, you should provide the means to retrieve passwords from your data store. The PasswordRecovery server control provides an easy way to accomplish this task.

Password recovery usually means sending the end user's password to him in an e-mail. Therefore, you need to set up an SMTP server (it might be the same as the application server). You configure for this server in the `web.config` file, as illustrated in Listing 16-22.

Listing 16-22: Configuring passwords to be sent via e-mail in the web.config file

```
<configuration>
   <system.web>
      <!-- Removed for clarity -->
   </system.web>

   <system.net>

      <mailSettings>
         <smtp from="someuser@email.com">
            <network host="localhost" port="25"
             defaultCredentials="true" />
         </smtp>
      </mailSettings>

   </system.net>
</configuration>
```

After you have the `<mailSettings>` element set up correctly, you can start to use the PasswordRecovery control. A simple use of the PasswordRecovery control is shown in Listing 16-23.

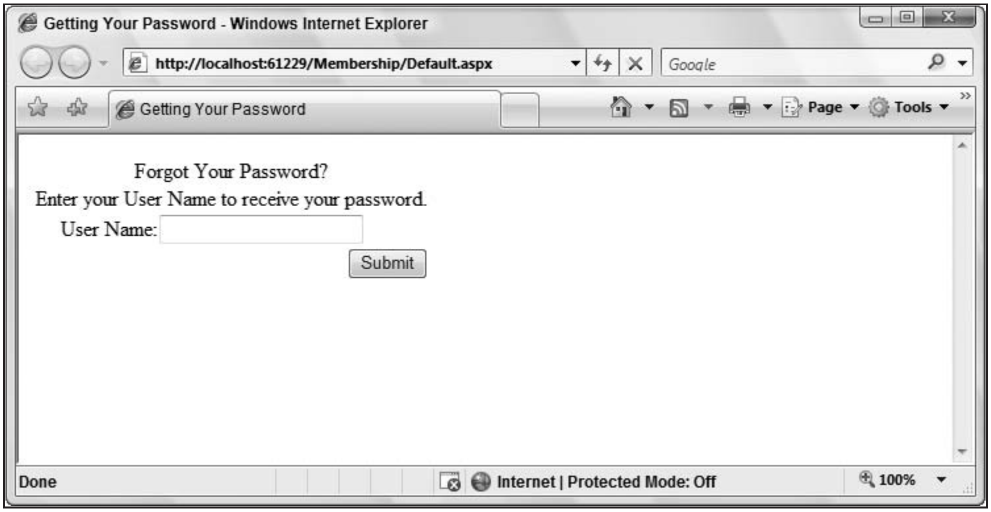
Listing 16-23: Using the PasswordRecovery control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Getting Your Password</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:PasswordRecovery ID="PasswordRecovery1" Runat="server">
            <MailDefinition From="someuser@email.com">
            </MailDefinition>
        </asp:PasswordRecovery>
    </form>
</body>
</html>
```

The `<asp:PasswordRecovery>` element needs a `<MailDefinition>` sub-element. The `<MailDefinition>` element contains details about the e-mail to be sent to the end user. The minimum requirement is that the `From` attribute is used, which provides the e-mail address for the From part of the e-mail. The `String` value of this attribute should be an e-mail address. Other attributes for the `<MailDefinition>` element include the following:

- ❑ `BodyFileName`
- ❑ `CC`
- ❑ `From`
- ❑ `IsBodyHtml`
- ❑ `Priority`
- ❑ `Subject`

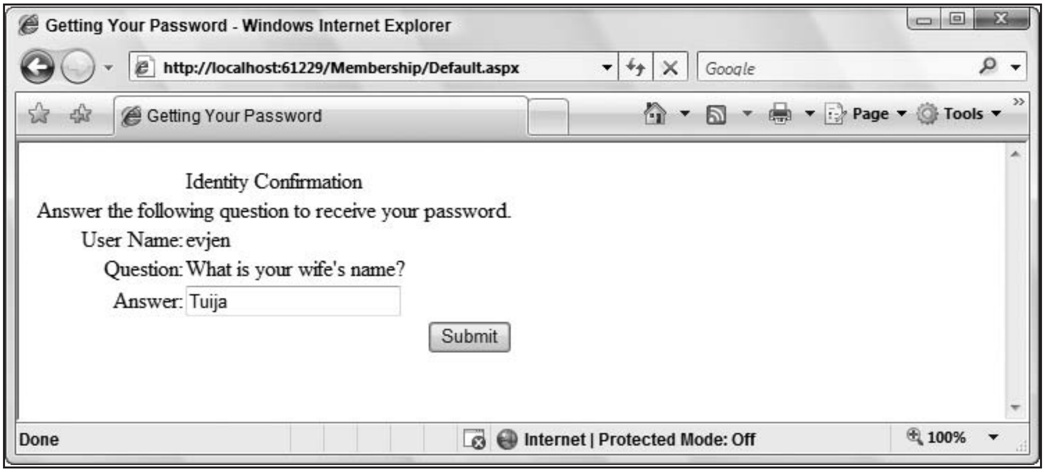
When you run this page, the PasswordRecovery control asks for the user's username, as shown in Figure 16-17.


Getting Your Password - Windows Internet Explorer
http://localhost:61229/Membership/Default.aspx
Google
Getting Your Password
Page
Tools
Done
Internet | Protected Mode: Off
100%

Forgot Your Password?
Enter your User Name to receive your password.
User Name:
Submit

Figure 16-17

When it has the username, the membership service retrieves the question and answer that was earlier entered by the end user and generates the view shown in Figure 16-18.


Getting Your Password - Windows Internet Explorer
http://localhost:61229/Membership/Default.aspx
Google
Getting Your Password
Page
Tools
Done
Internet | Protected Mode: Off
100%

Identity Confirmation
Answer the following question to receive your password.
User Name: evjen
Question: What is your wife's name?
Answer: Tuija
Submit

Figure 16-18

If the question is answered correctly (notice that the answer is case sensitive), an e-mail containing the password is generated and mailed to the end user. If the question is answered incorrectly, an error message is displayed. Of course, a question will not be used if you have the Question/Answer feature of the membership system disabled.

It is important to change some of your membership service settings in order for this entire process to work. At present, it will not work because of the way in which a user's password is hashed. The membership service data store is not storing the actual password — just a hashed version of it. Of course, it is useless for an end user to receive a hashed password.

In order for you to be able to send back an actual password to the user, you must change how the passwords are stored in the membership service data store. This is done (as stated earlier in the chapter) by changing the `PasswordFormat` attribute of your membership data provider. The other possible values (besides `Hashed`) are `Clear` and `Encrypted`. Changing the value to either `Clear` or `Encrypted` makes it possible for the passwords to be sent back to the end user in a readable format.

Generating Random Passwords

Certain applications must generate a random password when creating a user. In the days of ASP.NET 1.0/1.1, this was something you had to code yourself. ASP.NET 2.0 and 3.5, on the other hand, include a helper method that enables you to retrieve random passwords. Listing 16-24 shows an example of creating a helper method to pull a random password.

Listing 16-24: Generating a random password

VB

```
Protected Function GeneratePassword() As String
   Dim returnPassword As String
   returnPassword = Membership.GeneratePassword(10, 3)

   Return returnPassword
End Function
```

C#

```
protected string GeneratePassword()
{
   string returnPassword;
   returnPassword = Membership.GeneratePassword(10, 3);

   return returnPassword;
}
```

To generate a password randomly in ASP.NET 2.0 or 3.5, you can use the `GeneratePassword()` helper method. This method allows you to generate a random password of a specified length, and you can specify how many non-alphanumeric characters the password should contain (at minimum). This example utilizes this method five times to produce the results shown here (of course, your results will be different):

- `D](KQg6s2[`
- `$X.M9]*x2-`
- `Q+lIy2#zD%`
- `%kWZL@zy&f`
- `o]&IhL#iU1`

With your helper method in place, you can create users with random passwords, as shown in Listing 16-25.

Listing 16-25: Creating users with a random password

VB

```
Membership.CreateUser(TextBox1.Text, GeneratePassword().ToString())
```

C#

```
Membership.CreateUser(TextBox1.Text, GeneratePassword().ToString());
```

ASP.NET 3.5 Authorization

Now that you can deal with the registration and authentication of users who want to access your Web applications, the next step is authorization. What are they allowed to see and what roles do they take? These are important questions for any Web application. First, learn how to show only certain items to authenticated users while you show different items to unauthenticated users.

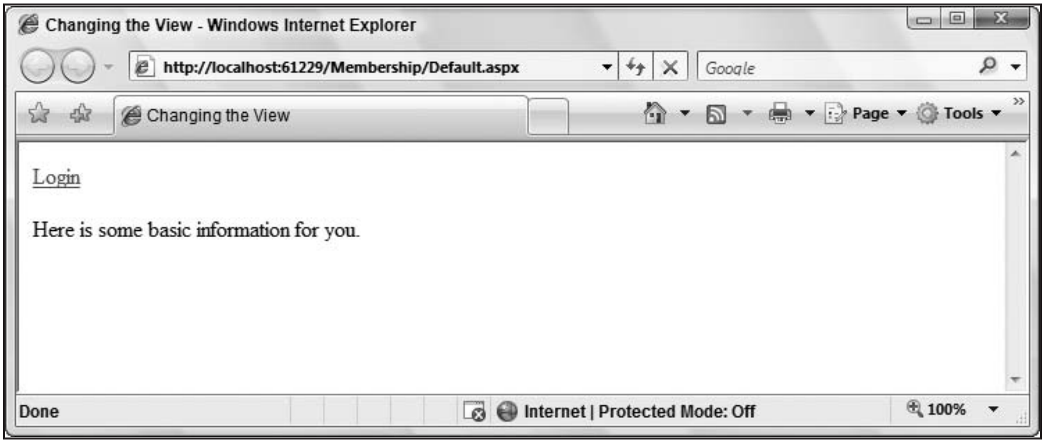
Using the LoginView Server Control

The LoginView server control allows you to control who views what information on a particular part of a page. Using the LoginView control, you can dictate which parts of the pages are for authenticated users and which parts of the pages are for unauthenticated users. Listing 16-26 shows an example of this control.

Listing 16-26: Controlling information viewed via the LoginView control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Changing the View</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:LoginStatus ID="LoginStatus1" Runat="server" />
        <p>
        <asp:LoginView ID="LoginView1" Runat="server">
            <LoggedInTemplate>
                Here is some REALLY important information that you should know
                about all those people that are not authenticated!
            </LoggedInTemplate>
            <AnonymousTemplate>
                Here is some basic information for you.
            </AnonymousTemplate>
        </asp:LoginView><p>
    </form>
</body>
</html>
```

The `<asp:LoginView>` control is a templated control that takes two possible subelements — the `<LoggedInTemplate>` and `<AnonymousTemplate>` elements. In this case, the information defined in the `<AnonymousTemplate>` section (see Figure 16-19) is for unauthenticated users.

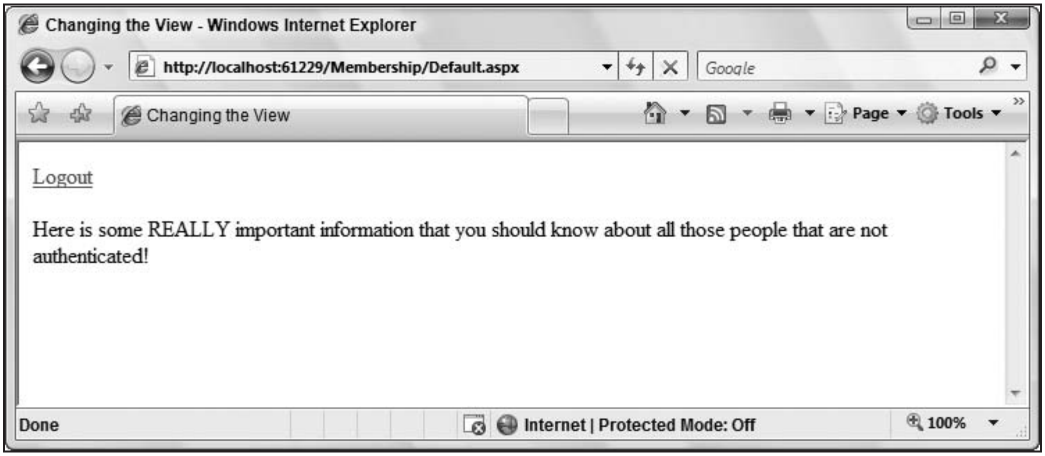

Changing the View - Windows Internet Explorer
http://localhost:61229/Membership/Default.aspx
Google
Changing the View
Page
Tools
Done
Internet | Protected Mode: Off
100%

Login

Here is some basic information for you.

Figure 16-19

It is quite different from what authenticated users see defined in the `<LoggedInTemplate>` section (see Figure 16-20).


Changing the View - Windows Internet Explorer
http://localhost:61229/Membership/Default.aspx
Google
Changing the View
Page
Tools
Done
Internet | Protected Mode: Off
100%

Logout

Here is some REALLY important information that you should know about all those people that are not authenticated!

Figure 16-20

Only simple ASCII text is placed inside both of these templates, but you can actually place anything else within the template including additional server controls. This means that you can show entire sections of pages, including forms, from within the templated sections.

Besides using just the `<LoggedInTemplate>` and the `<AnonymousTemplate>` of the LoginView control, you can also enable sections of a page or specific content for entities that are part of a particular role — such as someone who is part of the `Admin` group. You can accomplish this by using the `<RoleGroups>` section of the LoginView control, as shown in Listing 16-27.

Listing 16-27: Providing a view for a particular group

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Changing the View</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:LoginStatus ID="LoginStatus1" Runat="server" />
        <p>
        <asp:LoginView ID="LoginView1" Runat="server">
            <LoggedInTemplate>
                Here is some REALLY important information that you should know
                about all those people that are not authenticated!
            </LoggedInTemplate>
            <AnonymousTemplate>
                Here is some basic information for you.
            </AnonymousTemplate>
            <RoleGroups>
                <asp:RoleGroup Roles="Admins">
                    <ContentTemplate>
                       You are an Admin!
                    </ContentTemplate>
                </asp:RoleGroup>
                <asp:RoleGroup Roles="CoolPeople">
                    <ContentTemplate>
                       You are cool!
                    </ContentTemplate>
                </asp:RoleGroup>
            </RoleGroups>
        </asp:LoginView><p>
    </form>
</body>
</html>
```

To show content for a particular group of users, you add a `<RoleGroups>` element to the LoginView control. The `<RoleGroups>` section can take one or more RoleGroup controls (you will not find this control in Visual Studio's Toolbox). To provide content to display using the RoleGroup control, you provide a `<ContentTemplate>` element, which enables you to define the content to be displayed for an entity that belongs to the specified role. What is placed in the `<ContentTemplate>` section completely depends on you. You can place raw text (as shown in the example) or even other ASP.NET controls.

Be cautious of the order in which you place the defined roles in the `<RoleGroups>` section. When users log in to a site, they are first checked to see if they match one of the defined roles. The first (uppermost) role matched is the view used for the LoginView control — even if they match more than one role. You can also place more than one role in the `Roles` attribute of the `<asp:RoleGroups>` control, like this:

```
<asp:RoleGroup Roles="CoolPeople, HappyPeople">
   <ContentTemplate>
      You are cool or happy (or both)!
   </ContentTemplate>
</asp:RoleGroup>
```

Setting Up Your Web Site for Role Management

In addition to the membership service just reviewed, ASP.NET provides you with the other side of the end-user management service — the ASP.NET role management service. The membership service covers all the details of authentication for your applications, whereas the role management service covers authorization. Just as the membership service can use any of the data providers listed earlier, the role management service can also use a provider that is focused on SQL Server (SqlRoleProvider) or any custom providers. In fact, this service is comparable to the membership service in many ways. Figure 16-21 shows you a simple diagram that details some particulars of the role management service.

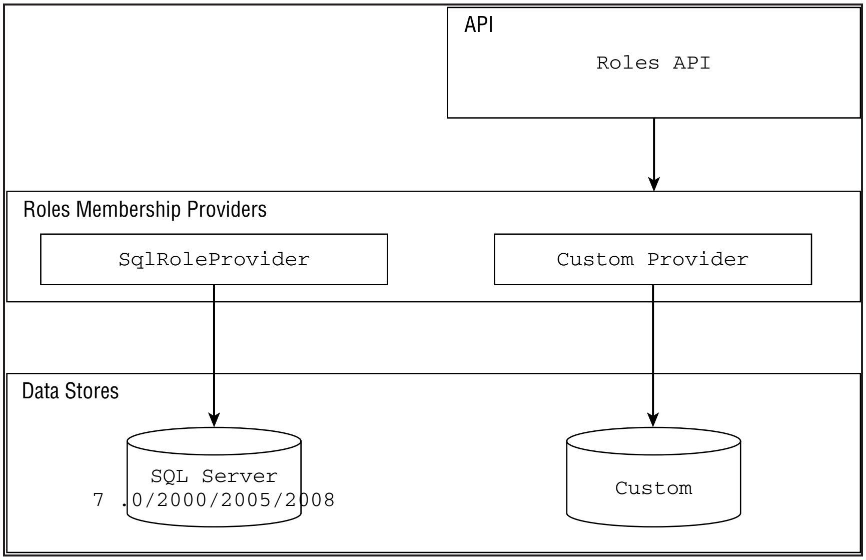


Figure 16-21

Making Changes to the <roleManager> Section

The first step in working with the role management service is to change any of the role management provider's behaviors either in the `machine.config` or from the `web.config` files. If you look in the `machine.config.comments` file, you will see an entire section that deals with the role management service (see Listing 16-28).

Listing 16-28: Role management provider settings in the machine.config.comments file

```
<roleManager
 enabled="false"
 cacheRolesInCookie="false"
 cookieName=".ASPXROLES"
 cookieTimeout="30"
 cookiePath="/"
 cookieRequireSSL="false"
 cookieSlidingExpiration="true"
```

```
    cookieProtection="All"
    defaultProvider="AspNetSqlRoleProvider"
    createPersistentCookie="false"
    maxCachedResults="25">
      <providers>
         <clear />
         <add connectionStringName="LocalSqlServer" applicationName="/"
          name="AspNetSqlRoleProvider" type="System.Web.Security.SqlRoleProvider,
          System.Web, Version=2.0.0.0, Culture=neutral,
          PublicKeyToken=b03f5f7f11d50a3a" />
         <add applicationName="/" name="AspNetWindowsTokenRoleProvider"
          type="System.Web.Security.WindowsTokenRoleProvider, System.Web,
          Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
      </providers>
   </roleManager>
```

The role management service defines its settings from within the `machine.config.comments` file, as shown in the previous code listing. You can make changes to these settings either directly in the `machine.config` file or by overriding any of the higher level settings you might have by making changes in the `web.config` file (thereby making changes only to the application at hand).

The main settings are defined in the `<roleManager>` element. Some of the attributes of the `<roleManager>` element are defined in the following table.

Attribute	**Description**
`enabled`	Defines whether the role management service is enabled for the application. This attribute takes a `Boolean` value and is set to `False` by default. This means that the role management service is disabled by default. This is done to avoid breaking changes that would occur for users migrating from ASP.NET 1.0/1.1 to ASP.NET 2.0 or 3.5. Therefore, you must first change this value to `True` in either the `machine.config` or the `web.config` file.
`cacheRolesInCookie`	Defines whether the roles of the user can be stored within a cookie on the client machine. This attribute takes a `Boolean` value and is set to `True` by default. This is an ideal situation because retrieving the roles from the cookie prevents ASP.NET from looking up the roles of the user via the role management provider. Set it to `False` if you want the roles to be retrieved via the provider for all instances.
`cookieName`	Defines the name used for the cookie sent to the end user for role management information storage. By default, this cookie is named `.ASPXROLES`, and you probably will not change this.
`cookieTimeout`	Defines the amount of time (in minutes) after which the cookie expires. The default value is `30` minutes.
`cookieRequireSSL`	Defines whether you require that the role management information be sent over an encrypted wire (SSL) instead of being sent as clear text. The default value is `False`.

Attribute	Description
`cookieSlidingExpiration`	Specifies whether the timeout of the cookie is on a sliding scale. The default value is `True`. This means that the end user's cookie does not expire until 30 minutes (or the time specified in the `cookieTimeout` attribute) after the last request to the application has been made. If the value of the `cookieSlidingExpiration` attribute is set to `False`, the cookie expires 30 minutes from the first request.
`createPersistentCookie`	Specifies whether a cookie expires or if it remains alive indefinitely. The default setting is `False` because a persistent cookie is not always advisable for security reasons.
`cookieProtection`	Specifies the amount of protection you want to apply to the cookie stored on the end user's machine for management information. The possible settings include `All`, `None`, `Encryption`, and `Validation`. You should always attempt to use `All`.
`defaultProvider`	Defines the provider used for the role management service. By default, it is set to `AspNetSqlRoleProvider`.

Making Changes to the web.config File

The next step is to configure your `web.config` file so that it can work with the role management service. Certain pages or subsections of your application may be accessible only to people with specific roles. To manage this access, you define the access rights in the `web.config` file. The necessary changes are shown in Listing 16-29.

Listing 16-29: Changing the web.config file

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>

    <system.web>
        <roleManager enabled="true"/>
        <authentication mode="Forms" />
        <authorization>
           <deny users="?" />
        </authorization>
    </system.web>

    <location path="AdminPage.aspx">
        <system.web>
           <authorization>
              <allow roles="AdminPageRights" />
              <deny users="*" />
           </authorization>
        </system.web>
    </location>

</configuration>
```

This `web.config` file is doing a couple of things. First, the function of the first `< system.web >` section is no different from that of the membership service shown earlier in the chapter. The `<deny>` element is denying all unauthenticated users across the board.

The second section of this `web.config` file is rather interesting. The `<location>` element is used to define the access rights of a particular page in the application (`AdminPage.aspx`). In this case, only users contained in the `AdminPageRights` role are allowed to view the page, but all other users — regardless of whether they are authenticated — are not allowed to view the page. When using the asterisk (`*`) as a value of the `users` attribute of the `<deny>` element, you are saying that all users (regardless of whether they are authenticated) are not allowed to access the resource being defined. This overriding denial of access, however, is broken open a bit via the use of the `<allow>` element, which allows users contained within a specific role.

Adding and Retrieving Application Roles

Now that the `machine.config` or the `web.config` file is in place, you can add roles to the role management service. The role management service, just like the membership service, uses data stores to store information about the users. These examples focus primarily on using Microsoft SQL Server Express Edition as the provider because it is the default provider.

One big difference between the role management service and the membership service is that no server controls are used for the role management service. You manage the application's roles and the user's role details through a Roles API or through the Web Site Administration Tool provided with ASP.NET 3.5. Listing 16-30 shows how to use some of the new methods to add roles to the service.

Listing 16-30: Adding roles to the application

VB

```
<%@ Page Language="VB" %>

<script runat="server">
   Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
       If Not Page.IsPostBack Then
          ListBoxDataBind()
       End If
   End Sub

   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
       Roles.CreateRole(TextBox1.Text)
       ListBoxDataBind()
   End Sub

   Protected Sub ListBoxDataBind()
       ListBox1.DataSource = Roles.GetAllRoles()
       ListBox1.DataBind()
   End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Role Manager</title>
```

Continued

```
</head>
<body>
    <form id="form1" runat="server">
        <h1>Role Manager</h1>
        Add Role:<br />
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
        <p><asp:Button ID="Button1" Runat="server" Text="Add Role to Application"
            OnClick="Button1_Click" /></p>
        Roles Defined:<br />
        <asp:ListBox ID="ListBox1" Runat="server">
        </asp:ListBox>
    </form>
</body>
</html>
```

C#

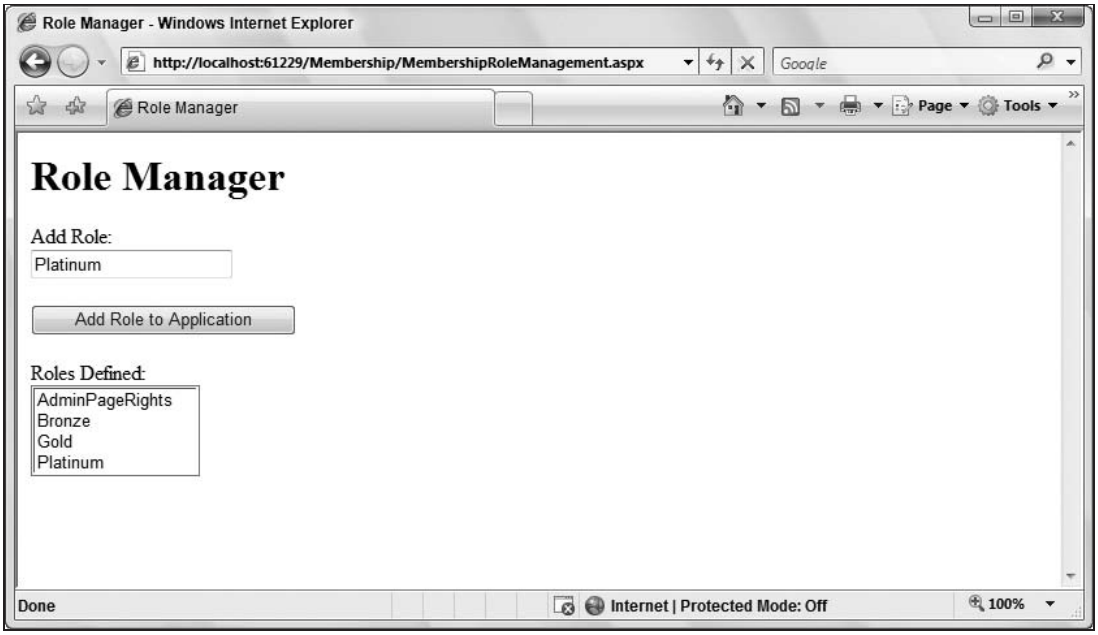
```
<%@ Page Language="C#" %>

<script runat="server">
   protected void Page_Load(object sender, EventArgs e)
   {
      if (!Page.IsPostBack)
      {
         ListBoxDataBind();
      }
   }

   protected void Button1_Click(object sender, EventArgs e)
   {
      Roles.CreateRole(TextBox1.Text.ToString());
      ListBoxDataBind();
   }

   protected void ListBoxDataBind()
   {
      ListBox1.DataSource = Roles.GetAllRoles();
      ListBox1.DataBind();
   }
</script>
```

This example enables you to enter roles into the text box and then to submit them to the role management service. The roles contained in the role management service are then displayed in the list box, as illustrated in Figure 16-22.

To enter the roles into the management service, you simply use the `CreateRole()` method of the `Roles` class. As with the `Membership` class, you do not instantiate the `Roles` class. To add roles to the role management service, use the `CreateRole()` method that takes only a single parameter — the name of the role as a `String` value:

```
Roles.CreateRole(rolename As String)
```

With this method, you can create as many roles as you want, but each role must be unique — otherwise an exception is thrown.



Role Manager - Windows Internet Explorer
http://localhost:61229/Membership/MembershipRoleManagement.aspx
Google
Role Manager
Page
Tools
Done
Internet | Protected Mode: Off
100%

Role Manager

Add Role:

Platinum

Add Role to Application

Roles Defined:

AdminPageRights
Bronze
Gold
Platinum

Figure 16-22

To retrieve the roles that are in the application's role management service (such as the list of roles displayed in the list box from the earlier example), you use the `GetAllRoles()` method of the `Roles` class. This method returns a `String` collection of all the available roles in the service:

```
Roles.GetAllRoles()
```

Deleting Roles

It would be just great to sit and add roles to the service all day long. Every now and then, however, you might want to delete roles from the service as well. Deleting roles is just as easy as adding roles to the role management service. To delete a role, you use one of the `DeleteRole()` method signatures. The first option of the `DeleteRole()` method takes a single parameter — the name of the role as a `String` value. The second option takes the name of the role plus a `Boolean` value that determines whether to throw an exception when one or more members are contained within that particular role (so that you don't accidentally delete a role with users in it when you don't mean to):

```
Roles.DeleteRole(rolename As String)

Roles.DeleteRole(rolename As String, throwOnPopulatedRole As Boolean)
```

Listing 16-31 is a partial code example that builds on Listing 16-30. For this example, add an additional button, which initiates a second button-click event that deletes the role from the service.

Listing 16-31: Deleting roles from the application

VB

```
Protected Sub DeleteButton_Click(ByVal sender As Object, _
   ByVal e As System.EventArgs)

   For Each li As ListItem In ListBox1.Items
      If li.Selected = True Then
         Roles.DeleteRole(li.ToString())
      End If
   Next

   ListBoxDataBind()
End Sub
```

C#

```
protected void DeleteButton_Click(object sender, EventArgs e)
{
   foreach (ListItem li in ListBox1.Items) {
      if (li.Selected == true) {
         Roles.DeleteRole(li.ToString());
      }
   }

   ListBoxDataBind();
}
```

This example deletes the selected items from the ListBox control. If more than one selection is made (meaning that you have placed the attribute `SelectionMode = "Multiple"` in the ListBox control), each of the roles is deleted from the service, in turn, in the `For Each` loop. Although `Roles.DeleteRole(li.ToString())` is used to delete the role, `Roles.DeleteRole(li.ToString(), True)` could also be used to make sure that no roles are deleted if that role contains any members.

Adding Users to Roles

Now that the roles are in place and it is possible to delete these roles if required, the next step is adding users to the roles created. A role does not do much good if no users are associated with it. To add a single user to a single role, you use the following construct:

```
Roles.AddUserToRole(username As String, rolename As String)
```

To add a single user to multiple roles at the same time, you use this construct:

```
Roles.AddUserToRoles(username As String, rolenames() As String)
```

To add multiple users to a single role, you use the following construct:

```
Roles.AddUsersToRole(usernames() As String, rolename As String)
```

Then, finally, to add multiple users to multiple roles, you use the following construct:

```
Roles.AddUsersToRoles(usernames() As String, rolenames() As String)
```

The parameters that can take collections, whether they are `usernames()` or `rolenames()`, are presented to the method as `String` arrays.

Getting All the Users of a Particular Role

Looking up information is easy in the role management service, whether you are determining which users are contained within a particular role or whether you want to know the roles that a particular user belongs to.

Methods are available for either of these scenarios. First, look at how to determine all the users contained in a particular role, as illustrated in Listing 16-32.

Listing 16-32: Looking up users in a particular role

VB

```
<%@ Page Language="VB" %>

<script runat="server">
   Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
       If Not Page.IsPostBack Then
          DropDownDataBind()
       End If
   End Sub

   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
       GridView1.DataSource = Roles.GetUsersInRole(DropDownList1.SelectedValue)
       GridView1.DataBind()
       DropDownDataBind()
   End Sub

   Protected Sub DropDownDataBind()
      DropDownList1.DataSource = Roles.GetAllRoles()
      DropDownList1.DataBind()
   End Sub

</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Role Manager</title>
</head>
<body>
    <form id="form1" runat="server">
        Roles:
        <asp:DropDownList ID="DropDownList1" Runat="server">
        </asp:DropDownList>
        <asp:Button ID="Button1" Runat="server" Text="Get Users In Role"
         OnClick="Button1_Click" />
        <br />
        <br />
        <asp:GridView ID="GridView1" Runat="server">
        </asp:GridView>
    </form>
```

Continued

```
</body>
</html>
```

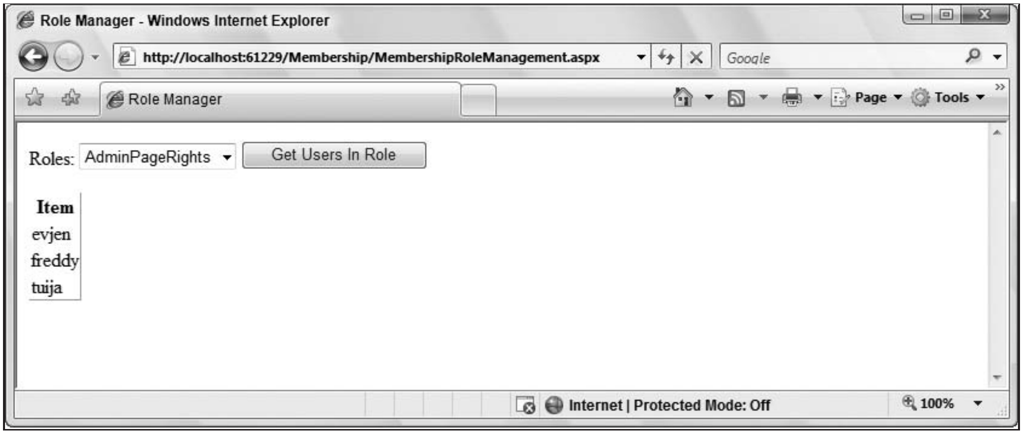
C#

```
<%@ Page Language="C#" %>

<script runat="server">
   protected void Page_Load(object sender, EventArgs e)
   {
      if (!Page.IsPostBack)
      {
         DropDownDataBind();
      }
   }

   protected void Button1_Click(object sender, EventArgs e)
   {
      GridView1.DataSource = Roles.GetUsersInRole(DropDownList1.SelectedValue);
      GridView1.DataBind();
      DropDownDataBind();
   }

   protected void DropDownDataBind()
   {
      DropDownList1.DataSource = Roles.GetAllRoles();
      DropDownList1.DataBind();
   }
</script>
```

This page creates a drop-down list that contains all the roles for the application. Clicking the button displays all the users for the selected role. Users of a particular role are determined using the `GetUsersInRole()` method. This method takes a single parameter — a `String` value representing the name of the role:

```
Roles.GetUsersInRole(rolename As String)
```

When run, the page looks similar to the page shown in Figure 16-23.


Role Manager - Windows Internet Explorer
http://localhost:61229/Membership/MembershipRoleManagement.aspx
Google
Role Manager
Page
Tools
Roles:
AdminPageRights
Get Users In Role
Internet | Protected Mode: Off
100%

Item
evjen
freddy
tuija

Figure 16-23

Getting All the Roles of a Particular User

To determine all the roles for a particular user, create a page with a single text box and a button. In the text box, you type the name of the user; and a button click initiates the retrieval and populates a GridView control. The button click event (where all the action is) is illustrated in Listing 16-33.

Listing 16-33: Getting all the roles of a specific user

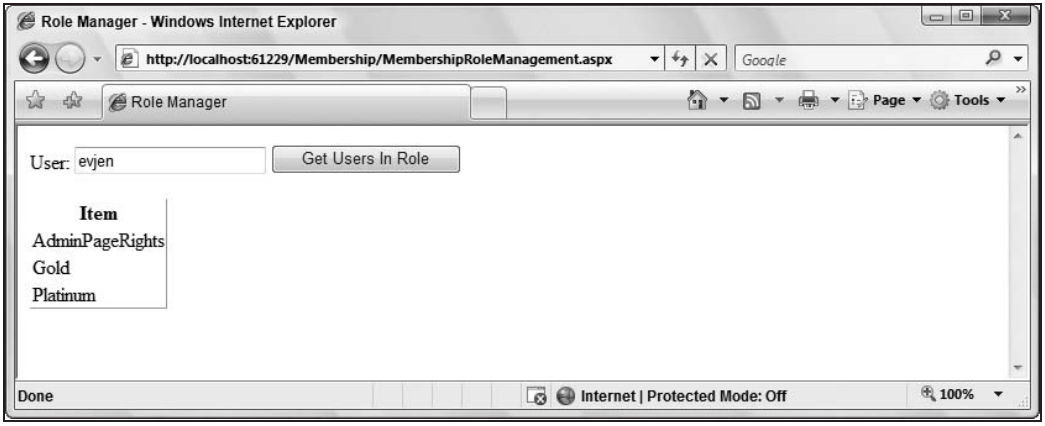
VB

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
   GridView1.DataSource = Roles.GetRolesForUser(TextBox1.Text)
   GridView1.DataBind()
End Sub
```

C#

```
protected void Button1_Click(object sender, EventArgs e)
{
   GridView1.DataSource = Roles.GetRolesForUser(TextBox1.Text.ToString());
   GridView1.DataBind();
}
```

The preceding code produces something similar to what is shown in Figure 16-24.


Role Manager - Windows Internet Explorer
http://localhost:61229/Membership/MembershipRoleManagement.aspx
Google
Role Manager
Page
Tools
Get Users In Role
Done
Internet | Protected Mode: Off
100%

User: evjen

Item
AdminPageRights
Gold
Platinum

Figure 16-24

To get the roles of a particular user, you simply use the `GetRolesForUser()` method. This method has two possible signatures. The first is shown in the preceding example — a `String` value that represents the name of the user. The other option is an invocation of the method without any parameters listed. This returns the roles of the user who has logged in to the membership service.

Removing Users from Roles

In addition to adding users to roles, you can also easily remove users from roles. To delete or remove a single user from a single role, you use the following construct:

```
Roles.RemoveUserFromRole(username As String, rolename As String)
```

To remove a single user from multiple roles at the same time, you use this construct:

```
Roles.RemoveUserFromRoles(username As String, rolenames() As String)
```

To remove multiple users from a single role, you use the following construct:

```
Roles.RemoveUsersFromRole(usernames() As String, rolename As String)
```

Then, finally, to remove multiple users from multiple roles, you use the following construct:

```
Roles.RemoveUsersFromRoles(usernames() As String, rolenames() As String)
```

The parameters shown as collections, whether they are `usernames()` or `rolenames()`, are presented to the method as `String` arrays.

Checking Users in Roles

One final action you can take is checking whether a particular user is in a role. You can go about this in a couple of ways. The first is using the `IsUserInRole()` method.

The `IsUserInRole()` method takes two parameters — the username and the name of the role:

```
Roles.IsUserInRole(username As String, rolename As String)
```

This method returns a `Boolean` value on the status of the user, and it can be used as shown in Listing 16-34.

Listing 16-34: Checking a user's role status

VB

```
If (Roles.IsUserInRole(TextBox1.Text, "AdminPageRights")) Then
   ' perform action here
End If
```

C#

```
if (Roles.IsUserInRole(TextBox1.Text.ToString(), "AdminPageRights"))
{
   // perform action here
}
```

The other option, in addition to the `IsUserInRole()` method, is to use `FindUsersInRole()`. This method enables you make a name search against all the users in a particular role. The `FindUsersInRole()` method takes two parameters — the name of the role and the username, both as `String` values:

```
Roles.FindUsersInRole(rolename As String, username As String)
```

Listing 16-35 shows an example of this method.

Listing 16-35: Checking for a specific user in a particular role

VB

```
<%@ Page Language="VB" %>

<script runat="server">
   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
       GridView1.DataSource = _
          Roles.FindUsersInRole("AdminPageRights", TextBox1.Text)
       GridView1.DataBind()
   End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Role Manager</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
        <asp:Button ID="Button1" Runat="server" Text="Button"
         OnClick="Button1_Click" />
        <p><asp:GridView ID="GridView1" Runat="server">
        </asp:GridView></p>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
   protected void Button1_Click(object sender, EventArgs e)
   {
        GridView1.DataSource =
           Roles.FindUsersInRole("AdminPageRights", TextBox1.Text.ToString());
        GridView1.DataBind();
   }
</script>
```

Understanding How Roles Are Cached

By default, after you retrieve a user's roles from the data store underlying the role management service, you can store these roles as a cookie on the client machine. This is done so you do not have to access the data store each and every time the application needs a user's role status. There is always a bit of risk in working with cookies because the end user can manipulate the cookie and thereby gain access to information or parts of an application that normally would be forbidden to that particular user.

Although roles are cached in a cookie, the default is that they are cached for only 30 minutes at a time. You can deal with this role cookie in several ways — some of which might help to protect your application better.

One protection for your application is to delete this role cookie, using the `DeleteCookie()` method of the Roles API, when the end user logs on to the site. This is illustrated in Listing 16-36.

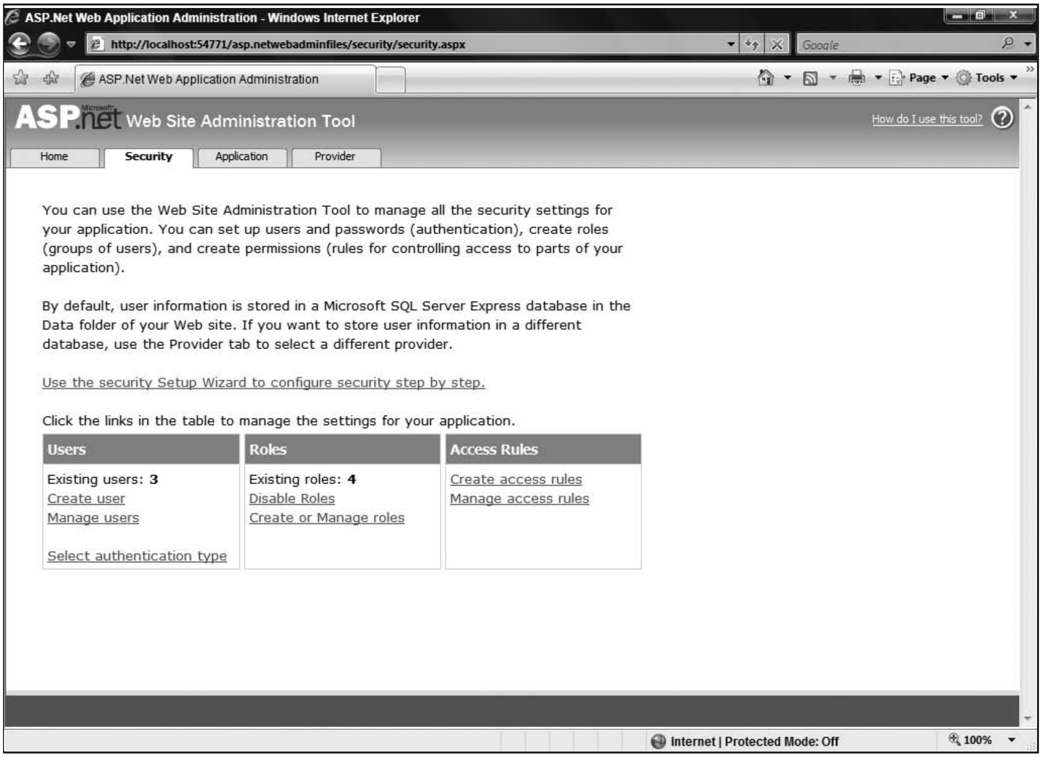
Listing 16-36: Deleting the end user's role cookie upon authentication

VB

```
If Membership.ValidateUser(TextBox1.Text, TextBox2.Text) Then
   Roles.DeleteCookie()
   FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, False)
Else
   Label1.Text = "You are not registered with the site."
End If
```

C#

```
if (Membership.ValidateUser(TextBox1.Text.ToString(), TextBox2.Text.ToString()) {
   Roles.DeleteCookie();
   FormsAuthentication.RedirectFromLoginPage(TextBox1.Text.ToString(), false);
}
else {
   Label1.Text = "You are not registered with the site.";
}
```


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:54771/asp.netwebadminfiles/security/security.aspx
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Internet | Protected Mode: Off
100%

You can use the Web Site Administration Tool to manage all the security settings for your application. You can set up users and passwords (authentication), create roles (groups of users), and create permissions (rules for controlling access to parts of your application).

By default, user information is stored in a Microsoft SQL Server Express database in the Data folder of your Web site. If you want to store user information in a different database, use the Provider tab to select a different provider.

Use the security Setup Wizard to configure security step by step.

Click the links in the table to manage the settings for your application.

Users	Roles	Access Rules
Existing users: **3** Create user Manage users Select authentication type	Existing roles: **4** Disable Roles Create or Manage roles	Create access rules Manage access rules

Figure 16-25

Using `Roles.DeleteCookie()` does exactly what you would think — it deletes from the client machine any cookie that is used to define the user's roles. If the end user is re-logging into the site, no problem should arise with re-authenticating his exact roles within the application. There is no need to rely on the contents of the cookie. This step provides a little more protection for your site.

Using the Web Site Administration Tool

Many of the actions shown in this chapter can also be performed through the Web Site Administration Tool shown in Figure 16-25. You can get at the ASP.NET Web Site Administration Tool by selecting Website ➪ ASP.NET Configuration from the Visual Studio 2008 menu.

Although you can easily use this tool to perform all the actions for you, often you perform these actions through your own applications as well. It is important to know all the possibilities when programming an ASP.NET application.

The Web Site Administration Tool is detailed in Chapter 34.

Public Methods of the Membership API

The public methods of the Membership API are detailed in the following table. You would use this API when working with the authentication process of your application.

Membership Methods	**Description**
`CreateUser`	Adds a new user to the appointed data store.
`DeleteUser`	Deletes a specified user from the data store.
`FindUsersByEmail`	Returns a collection of users who have an e-mail address to match the one provided.
`FindUsersByName`	Returns a collection of users who have a username to match the one provided.
`GeneratePassword`	Generates a random password of a length that you specify.
`GetAllUsers`	Returns a collection of all the users contained in the data store.
`GetNumberOfUsersOnline`	Returns an Integer that specifies the number of users who have logged in to the application. The time window during which users are counted is specified in the `machine.config` or the `web.config` files.
`GetUser`	Returns information about a particular user from the data store.
`GetUserNameByEmail`	Retrieves a username of a specific record from the data store based on an e-mail address search.
`UpdateUser`	Updates a particular user's information in the data store.
`ValidateUser`	Returns a Boolean value indicating whether a specified set of credentials is valid.

Public Methods of the Roles API

The public methods of the Roles API are detailed in the following table. You would use this API when working with the authorization process of your application.

Roles Methods	Description
`AddUsersToRole`	Adds a collection of users to a specific role.
`AddUsersToRoles`	Adds a collection of users to a collection of roles.
`AddUserToRole`	Adds a specific user to a specific role.
`AddUserToRoles`	Adds a specific user to a collection of roles.
`CreateRole`	Adds a new role to the appointed data store.
`DeleteCookie`	Deletes the cookie on the client used to store the roles to which the user belongs.
`DeleteRole`	Deletes a specific role in the data store. Using the proper parameters for this method, you can also control if roles are deleted or kept intact whether or not that particular role contains users.
`FindUsersInRole`	Returns a collection of users who have a username to match the one provided.
`GetAllRoles`	Returns a collection of all the roles stored in the data store.
`GetRolesForUser`	Returns a collection of roles for a specific user.
`IsUserInRole`	Returns a Boolean value that specifies whether a user is contained in a particular role.
`RemoveUserFromRole`	Removes a specific user from a specific role.
`RemoveUserFromRoles`	Removes a specific user from a collection of roles.
`RemoveUsersFromRole`	Removes a collection of users from a specific role.
`RemoveUsersFromRoles`	Removes a collection of users from a collection of roles.
`RoleExists`	Returns a Boolean value indicating whether a role exists in the data store.

Summary

This chapter covered two outstanding features available to ASP.NET 3.5. The membership and role management services that are now a part of ASP.NET make managing users and their roles almost trivial.

This chapter reviewed both the Membership and Roles APIs and the controls that also utilize these APIs. These controls and APIs follow the same data provider models as the rest of ASP.NET. The examples were presented using Microsoft SQL Server Express Edition for the back-end storage, but you can easily configure these systems to work with another type of data store.

17

Portal Frameworks and Web Parts

Internet and intranet applications have changed considerably since their introduction in the 1990s. Today's applications do not simply display the same canned information to every viewer; they do much more. Because of the wealth of information being exposed to end users, Internet and intranet applications must integrate large amounts of customization and personalization into their offerings.

Web sites that provide a plethora of offerings give end users the option to choose which parts of the site they want to view and which parts they want to hide. Ideally, end users can personalize the pages, deciding for themselves the order in which the content appears on the page. They should be able to move items around on the page as if it were a design surface.

In this situation, after pages are customized and established, end users need the capability to export their final page settings for storage. You certainly would not want an end user who has highly customized a page or a series of pages in your portal to be forced to reapply the settings each time he visits the site. Instead, you want to retain these setting points by moving them to a data store for later exposure.

Adding this kind of functionality is *expensive* — expensive in the sense that it can take a considerable amount of work on the part of the developer. Until ASP.NET 2.0, the developer had to build a personalization framework to be used by each page requiring the functionality. This type of work is error prone and difficult to achieve, which is why in most cases it was not done.

But wait. . .

Introducing Web Parts

To make it easier to retain the page customization settings that your end users apply to your page, Microsoft includes Web Parts as part of ASP.NET. Web Parts, part of the larger Portal Framework, provide an outstanding way to build a modular Web site that can be customized with dynamically reapplied settings on a per-user basis. Web Parts are objects in the Portal Framework which the end user can open, close, minimize, maximize, or move from one part of the page to another.

The Portal Framework enables you to build pages that contain multiple Web Parts — which are part of the ASP.NET server control framework and are used like any other ASP.NET server controls. This means that you can also extend Web Parts if necessary.

The components of the Portal Framework provide the means to build a truly dynamic Web site, whether that site is a traditional Internet site, an intranet site, a browser-based application, or any other typical portal.

When you first look at Web Parts in ASP.NET 3.5, it may remind you of Microsoft's SharePoint offering. Be forewarned, however, that these two technologies are not the same. Web Parts and the resulting Portal Framework, besides being offered in ASP.NET, are also used by the Windows SharePoint Services (WSS). Microsoft, as it often does, is simply creating singular technologies that can be used by other Microsoft offerings. In this process, Microsoft is trying to reach the Holy Grail of computing — *code reuse!*

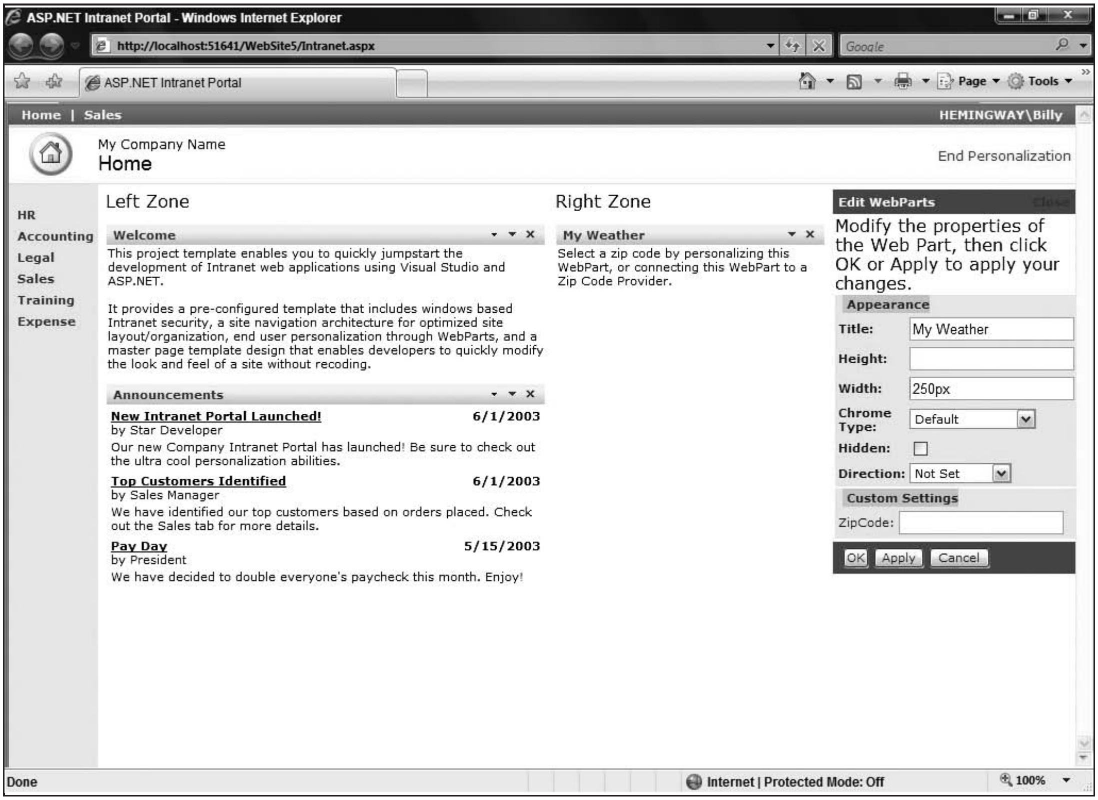
The modular and customizable sites that you can build with the Portal Framework enable you to place the Web page in view into several possible modes for the end user. The following list describes each of these available modes and what each means to the end user viewing the page:

- **Normal Mode:** Puts the page in a normal state, which means that the end user cannot edit or move sections of the page. This is the mode used for standard page viewing.
- **Edit Mode:** Enables end users to select particular sections on the page for editing. The selected section allows all types of editing capabilities from changing the part's title, the part's color, or even setting custom properties — such as allowing the end user to specify his zip code to pull up a customized weather report.
- **Design Mode:** Enables end users to rearrange the order of the page's modular components. The end user can bring items higher or lower within a zone, delete items from a zone, or move items from one page zone to another.
- **Catalog Mode:** Displays a list of available sections (Web Parts) that can be placed in the page. Catalog mode also allows the end user to select in which zone on the page the items should appear.

Figure 17-1 shows a screenshot of a sample portal utilizing the Portal Framework with the Edit mode enabled.

The Portal Framework is a comprehensive and well-thought-out framework that enables you to incorporate everything you would normally include in your ASP.NET applications. You can apply security using either Windows Authentication or Forms Authentication just as you can with a standard ASP.NET page. This framework also enables you to leverage the other aspects of ASP.NET 3.5, such as applying role management, personalization, and membership features to any portal that you build.

To help you understand how to build your own application on top of the Portal Framework, this chapter begins with the creation of a simple page that makes use of this new framework's utilities.


ASP.NET Intranet Portal - Windows Internet Explorer
http://localhost:51641/WebSite5/Intranet.aspx
Google
ASP.NET Intranet Portal
Page
Tools
Home | Sales
HEMINGWAY\Billy
End Personalization
HR
Accounting
Legal
Sales
Training
Expense
Close
OK
Apply
Cancel
Done
Internet | Protected Mode: Off
100%

My Company Name
Home

Left Zone

Welcome

This project template enables you to quickly jumpstart the development of Intranet web applications using Visual Studio and ASP.NET.

It provides a pre-configured template that includes windows based Intranet security, a site navigation architecture for optimized site layout/organization, end user personalization through WebParts, and a master page template design that enables developers to quickly modify the look and feel of a site without recoding.

Announcements

New Intranet Portal Launched! 6/1/2003
by Star Developer
Our new Company Intranet Portal has launched! Be sure to check out the ultra cool personalization abilities.

Top Customers Identified 6/1/2003
by Sales Manager
We have identified our top customers based on orders placed. Check out the Sales tab for more details.

Pay Day 5/15/2003
by President
We have decided to double everyone's paycheck this month. Enjoy!

Right Zone

My Weather

Select a zip code by personalizing this WebPart, or connecting this WebPart to a Zip Code Provider.

Edit WebParts

Modify the properties of the Web Part, then click OK or Apply to apply your changes.

Appearance

Title: My Weather
Height:
Width: 250px
Chrome Type: Default
Hidden: ☐
Direction: Not Set

Custom Settings

ZipCode:

Figure 17-1

Building Dynamic and Modular Web Sites

As you begin using the Portal Framework to build Web sites, note that the framework defines everything in *zones*. There are zones for laying out as well as for editing content. The zones that a page might incorporate are managed by a Portal Framework manager. The Portal framework manager performs the management on your behalf, meaning that you do not have to manage them yourself in any fashion. This makes working with the Portal Framework a breeze.

This framework contains a lot of moving parts and these multiple pieces that are heavily dependent upon each other. For this reason, this section starts at the beginning by examining the Portal Framework manager control: WebPartManager.

Introducing the WebPartManager Control

The WebPartManager control is an ASP.NET server control that completely manages the state of the zones and the content placed in these zones on a per-user basis. This control, which has no visual aspect, can add and delete items contained within each zone of the page. The WebPartManager control can also manage the communications sometimes required between different elements contained in the zones. For example, you can pass a specific name/value pair from one item to another item within the same zone, or between items contained in entirely separate zones. The WebPartManager control provides the capabilities to make this communication happen.

The WebPartManager control must be in place on every page in your application that works with the Portal Framework. A single WebPartManager control does not manage an entire application; instead, it manages on a per-page basis.

> *You can also place a WebPartManager server control on the master page (if you are using one) to avoid having to place one on each and every content page.*

Listing 17-1 shows a WebPartManager control added to an ASP.NET page.

Listing 17-1: Adding a WebPartManager control to an ASP.NET page

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Web Parts Example</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:WebPartManager ID="Webpartmanager1" runat="server">
        </asp:WebPartManager>
    </form>
</body>
</html>
```

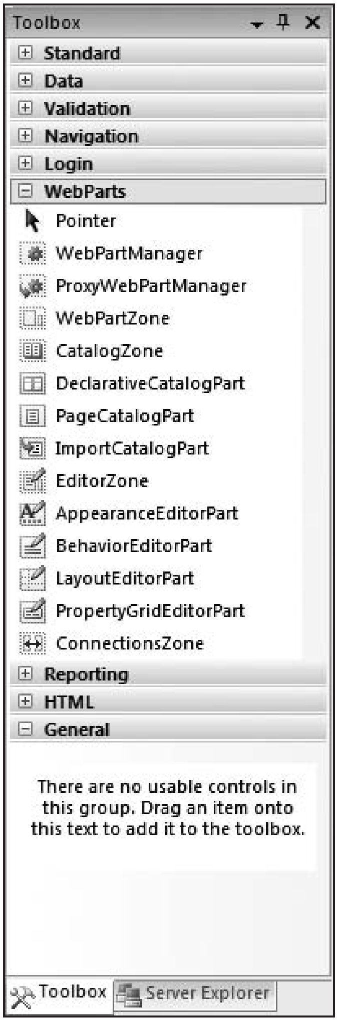
If you want to work from the design surface of Visual Studio 2008, you can drag and drop the WebPartManager control from the Toolbox to the design surface — but remember, it does not have a visual aspect and appears only as a gray box. You can find the WebPartManager control (and the other server controls that are part of the Portal Framework) in the WebParts section of the Toolbox, as shown in Figure 17-2.

Working with Zone Layouts

After you place the WebPartManager control on the page, the next step is to create zones from which you can utilize the Portal Framework. You should give this step some thought because it contributes directly to the usability of the page you are creating. Web pages are constructed in a linear fashion — either horizontally or vertically. Web pages are managed in square boxes — usually using tables that organize the columns and rows in which items appear on the page.

Web zones define specific rows or columns as individual content areas managed by the WebPartManager. For an example of a Web page that uses these zones, create a table similar to the one shown in Figure 17-3.

The black sections in Figure 17-3 will represent Web zones. The code used to produce the table with some basic controls in each of the zones is shown in Listing 17-2.


Toolbox
Standard
Data
Validation
Navigation
Login
WebParts
Pointer
WebPartManager
ProxyWebPartManager
WebPartZone
CatalogZone
DeclarativeCatalogPart
PageCatalogPart
ImportCatalogPart
EditorZone
AppearanceEditorPart
BehaviorEditorPart
LayoutEditorPart
PropertyGridEditorPart
ConnectionsZone
Reporting
HTML
General
Toolbox
Server Explorer

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

Figure 17-2

Listing 17-2: Creating multiple Web zones

```
<%@ Page Language="VB"%>
<%@ Register Src="DailyLinks.ascx" TagName="DailyLinks" TagPrefix="uc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Web Parts Example</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:WebPartManager ID="Webpartmanager1" runat="server">
```

Continued

```
        </asp:WebPartManager>
        <table cellpadding="5" border="1">
            <tr>
                <td colspan="3">
                    <h1>Bill Evjen's Web Page</h1>
                    <asp:WebPartZone ID="WebPartZone1" runat="server"
                     LayoutOrientation="Horizontal">
                        <ZoneTemplate>
                            <asp:Label ID="Label1" runat="server" Text="Label"
                             Title="Welcome to my web page!">
                             Welcome to the page!
                            </asp:Label>
                        </ZoneTemplate>
                    </asp:WebPartZone>
                </td>
            </tr>
            <tr valign="top">
                <td>
                    <asp:WebPartZone ID="WebPartZone2" runat="server">
                        <ZoneTemplate>
                            <asp:Image ID="Image1" runat="server"
                             ImageUrl="~/Images/Tuija.jpg" Width="150px"
                             Title="Tuija at the Museum">
                            </asp:Image>
                            <uc1:DailyLinks ID="DailyLinks1" runat="server"
                             Title="Daily Links">
                            </uc1:DailyLinks>
                        </ZoneTemplate>
                    </asp:WebPartZone>
                </td>
                <td>
                    <asp:WebPartZone ID="WebPartZone3" runat="server">
                        <ZoneTemplate>
                            <asp:Calendar ID="Calendar1" runat="server"
                             Title="Calendar">
                            </asp:Calendar>
                        </ZoneTemplate>
                    </asp:WebPartZone>
                </td>
                <td><!-- Blank for now -->
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

This page now has sections like the ones shown in Figure 17-3: a header section that runs horizontally and three vertical sections underneath the header. Running this page provides the result shown in Figure 17-4.

First, this page includes the `<asp:WebPartManager>` control that manages the items contained in the three zones on this page. Within the table, the `<asp:WebPartZone>` server control specifies three Web zones. You can declare each Web zone in one of two ways. You can use the `<asp:WebPartZone>` element directly in the code, or you can create the zones within the table by dragging and dropping WebPartZone controls onto the design surface at appropriate places within the table. In Figure 17-4, the table border width is intentionally turned on and set to `1` in order to show the location of the Web zones in greater detail. Figure 17-5 shows what the sample from Listing 17-2 looks like in the Design view of Visual Studio 2008.

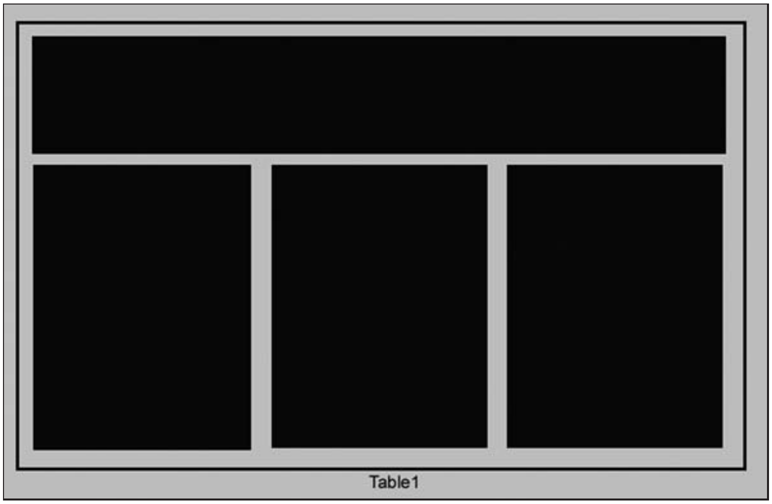


Figure 17-3

When using Visual Studio 2008, note that by default this IDE creates a Microsoft SQL Server Express Edition file called `ASPNETDB.MDF` and stores it in the `App_Data` folder of your Web Project. This database file is where the Portal Framework stores all the customization points.

> *Note that if you want this portal framework to run from SQL Server 7.0, 2000, 2005, or 2008, you should follow the se-up instructions that are defined in Chapter 12.*

Now that you have seen the use of WebPartZone controls, which are managed by the WebPartManager control, the next section takes a closer look at the WebPartZone server control itself.

Understanding the WebPartZone Control

The WebPartZone control defines an area of items, or Web Parts, that can be moved, minimized, maximized, deleted, or added based on programmatic code or user preferences. When you drag and drop WebPartZone controls onto the design surface using Visual Studio 2008, the WebPartZone control is drawn at the top of the zone, along with a visual representation of any of the items contained within the zone.

You can place almost anything in one of the Web zones. For example, you can include the following:

- ❑ HTML elements (when putting a `runat = "server"` on the element)
- ❑ HTML server controls

- ❑ Web server controls
- ❑ User controls
- ❑ Custom controls

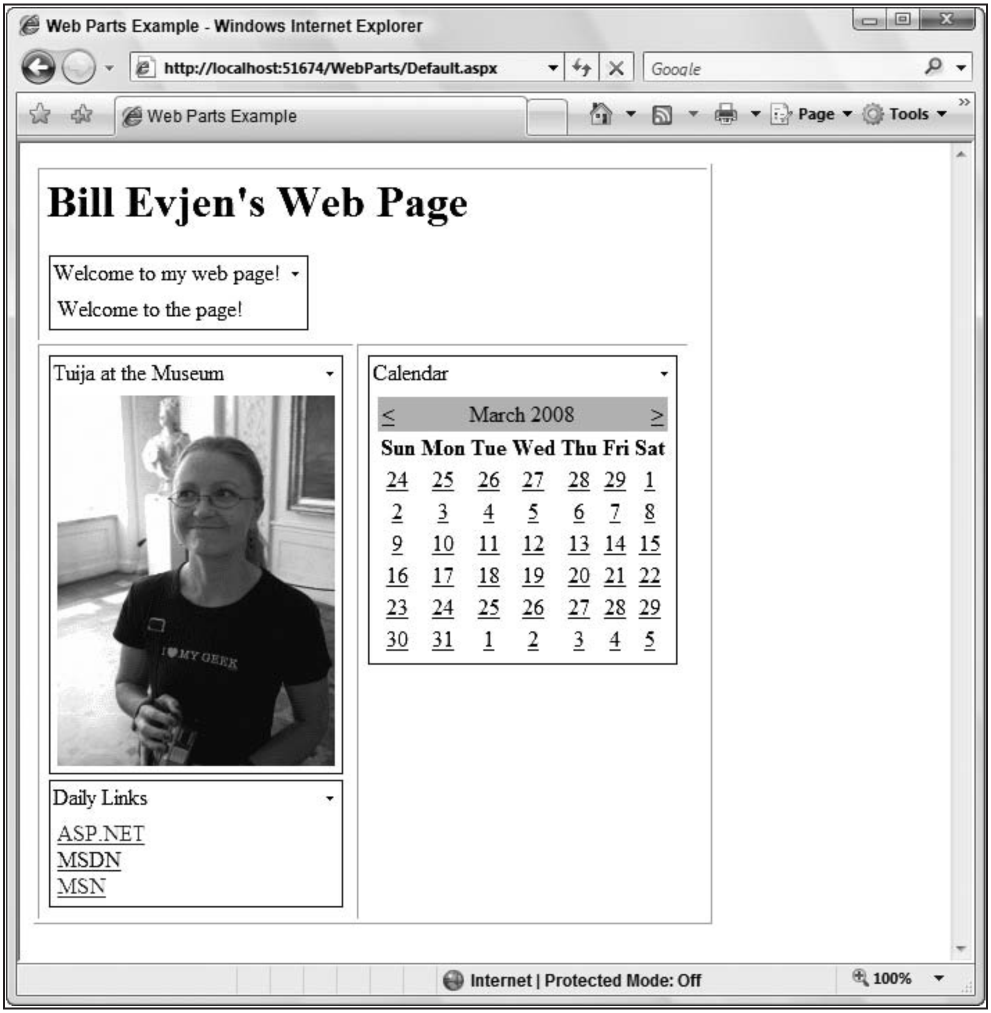

Web Parts Example - Windows Internet Explorer
http://localhost:51674/WebParts/Default.aspx
Google
Web Parts Example
Page
Tools
Bill Evjen's Web Page
Welcome to my web page!
Welcome to the page!
Tuija at the Museum
Calendar
March 2008
Sun Mon Tue Wed Thu Fri Sat
Daily Links
ASP.NET
MSDN
MSN
Internet | Protected Mode: Off
100%

Figure 17-4

WebPartZone controls are declared like this:

```
<asp:WebPartZone ID="WebPartZone1" Runat="server"></asp:WebPartZone>
```

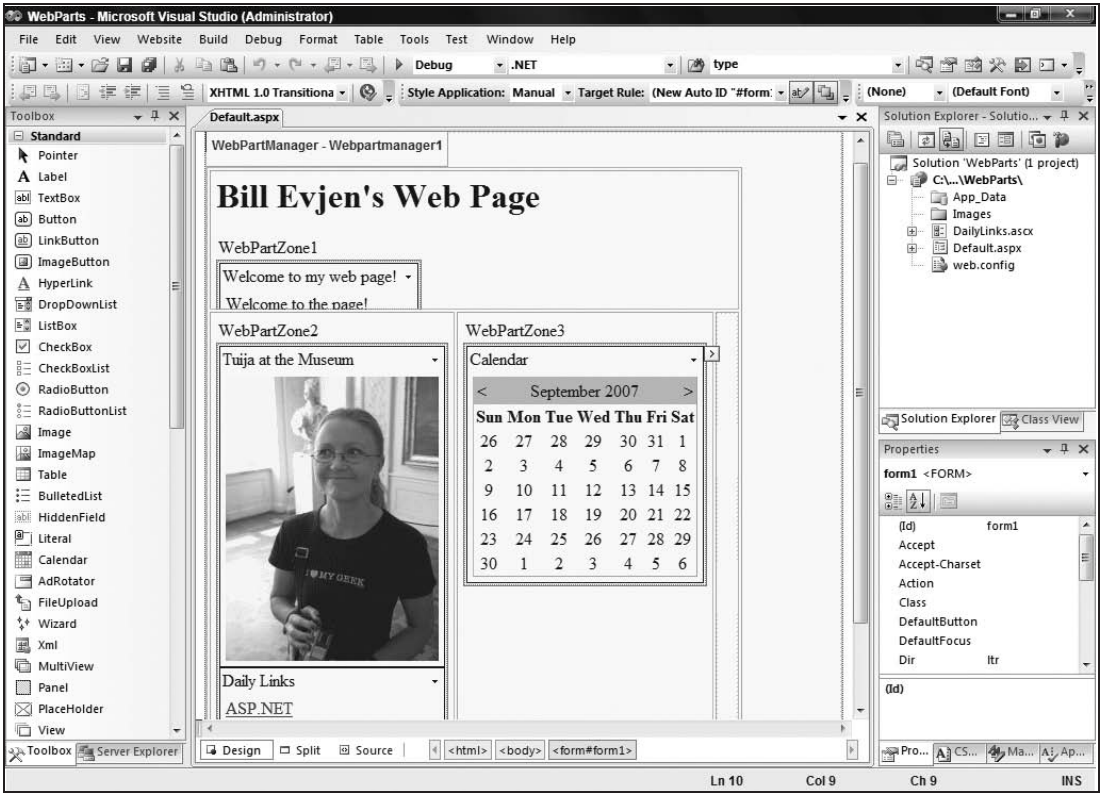

WebParts - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Format Table Tools Test Window Help
Debug
.NET
type
XHTML 1.0 Transitiona
Style Application: Manual
Target Rule: (New Auto ID "#form:
(None)
(Default Font)
Toolbox
Standard
Pointer
Label
TextBox
Button
LinkButton
ImageButton
HyperLink
DropDownList
ListBox
CheckBox
CheckBoxList
RadioButton
RadioButtonList
Image
ImageMap
Table
BulletedList
HiddenField
Literal
Calendar
AdRotator
FileUpload
Wizard
Xml
MultiView
Panel
PlaceHolder
View
Toolbox
Server Explorer
Default.aspx
WebPartManager - Webpartmanager1
Bill Evjen's Web Page
WebPartZone1
Welcome to my web page!
Welcome to the page!
WebPartZone2
Tuija at the Museum
Daily Links
ASP.NET
WebPartZone3
Calendar
September 2007
Sun Mon Tue Wed Thu Fri Sat
26 27 28 29 30 31 1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 1 2 3 4 5 6
Design
Split
Source
<html>
<body>
<form#form1>
Solution Explorer - Solutio...
Solution 'WebParts' (1 project)
C:\...\WebParts\
App_Data
Images
DailyLinks.ascx
Default.aspx
web.config
Solution Explorer
Class View
Properties
form1 <FORM>
(Id) form1
Accept
Accept-Charset
Action
Class
DefaultButton
DefaultFocus
Dir ltr
(Id)
Pro...
CS...
Ma...
Ap...
Ln 10
Col 9
Ch 9
INS

Figure 17-5

The LayoutOrientation Attribute

The Web Parts declared within a WebPartZone control can be displayed either horizontally or vertically. By default, all the items are displayed vertically, but to display the items horizontally, you simply add the `LayoutOrientation` attribute to the `<asp:WebPartZone>` element:

```
<asp:WebPartZone ID="WebPartZone1" Runat="server"
 LayoutOrientation="Horizontal"></asp:WebPartZone>
```

The first row in the table from Listing 17-2 uses horizontal orientation, whereas the other two zones use the default vertical orientation.

The ZoneTemplate Element

In order to include items within the templated WebPartZone control, you must include a `<ZoneTemplate>` element.

The `ZoneTemplate` element encapsulates all the items contained within a particular zone. The order in which they are listed in the `ZoneTemplate` section is the order in which they appear in the browser until changed by the end user or by programmatic code. The sample `<ZoneTemplate>` section used earlier is illustrated here:

```
<asp:WebPartZone ID="WebPartZone2" Runat="server">
   <ZoneTemplate>
      <asp:Image ID="Image1" Runat="server"
       ImageUrl="~/Images/Tuija.jpg" Width="150" Title="Tuija at the Museum">
      </asp:Image>
      <uc1:DailyLinks ID="DailyLinks1" runat="server" Title="Daily Links">
      </uc1:DailyLinks>
   </ZoneTemplate>
</asp:WebPartZone>
```

This zone contains two items — an Image server control and a user control consisting of a collection of links that come from an XML file.

Default Web Part Control Elements

By default, when you generate a page using the code from Listing 17-2, you discover that you can exert only minimal control over the Web Parts themselves. In the default view, which is not the most artistic in this case, you are able only to minimize or close a Web Part. You can see these options when you click on the down arrow that is presented next to the name of the Web Part.

Figure 17-6 shows what the Web Part that contains the Calendar control looks like after you minimize it. Notice also that if you opt to close one of the Web Parts, the item completely disappears. There seems to be no way to make it come back — even if you shut down the page and restart it. This is by design — so don't worry. I will show you how to get it back!

A few of the items included in the zones have new titles. By default, the title that appears at the top of the Web Part is the name of the control. For instance, you can see that the Calendar control is simply titled Calendar. If you add a Button control or any other control to the zone, at first it is simply titled Untitled. To give better and more meaningful names to the Web Parts that appear in a zone, you simply add a `Title` attribute to the control — just as was done with the Image control and the User control, which both appear on the page. In the preceding code example, the Image control is renamed to `Tuija at the Museum`, and the user control is given the `Title` value `Daily Links`.

Besides this little bit of default functionality, you can do considerably more with the Web Parts contained within this page, but you have to make some other additions. These are reviewed next.

Allowing the User to Change the Mode of the Page

Working with the `WebPartManager` class either directly or through the use of the WebPartManager server control, you can have the mode of the page changed. Changing the mode of the page being viewed allows the user to add, move, or change the pages they are working with. The nice thing about the Web Part capabilities of ASP.NET is that these changes are then recorded to the `ASPNETDB.MDF` database file and are, therefore, re-created the next time the user visits the page.

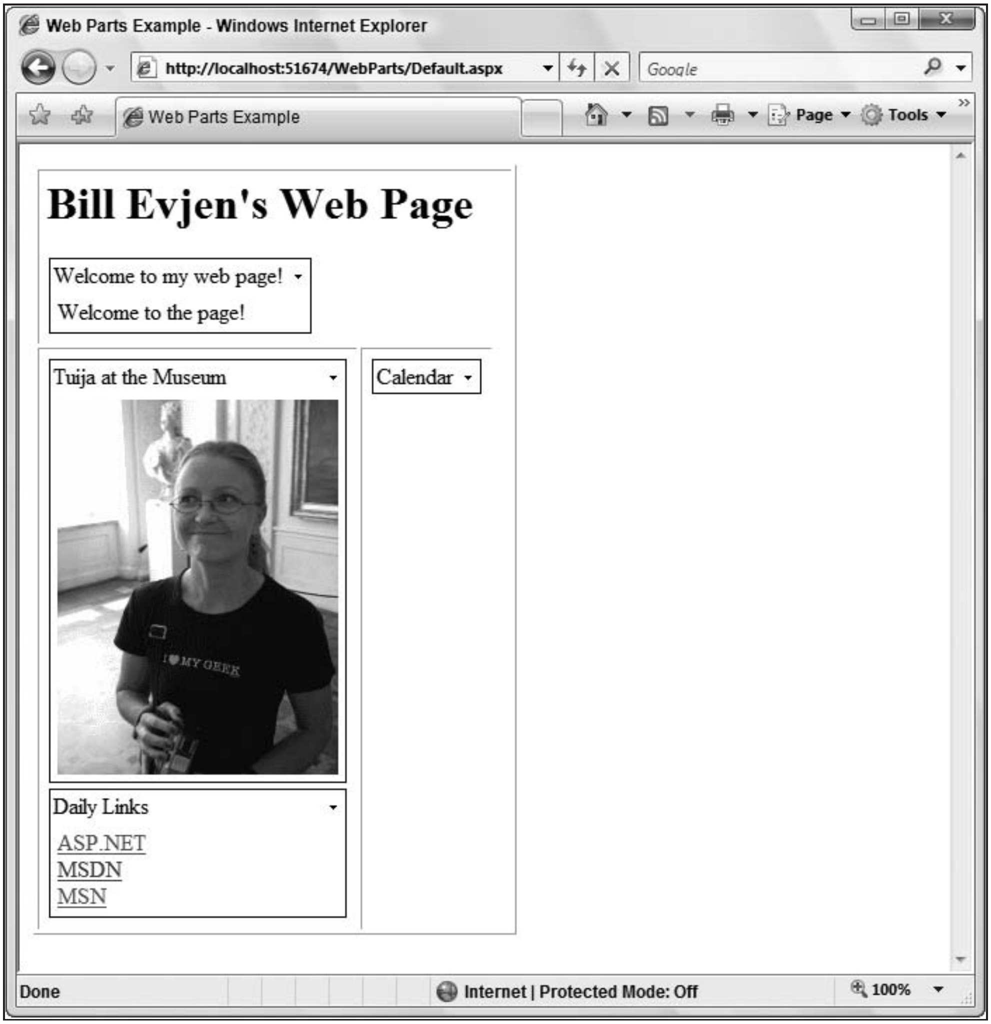

Web Parts Example - Windows Internet Explorer
http://localhost:51674/WebParts/Default.aspx
Google
Web Parts Example
Page
Tools
Bill Evjen's Web Page
Welcome to my web page!
Welcome to the page!
Tuija at the Museum
Calendar
Daily Links
ASP.NET
MSDN
MSN
Done
Internet | Protected Mode: Off
100%

Figure 17-6

Using the `WebPartManager` object, you can enable the user to do the following, as defined in this list:

- **Add new Web Parts to the page:** Includes Web Parts not displayed on the page by default and Web Parts that the end user has previously deleted. This aspect of the control works with the catalog capabilities of the Portal Framework, which is discussed shortly.
- **Enter the Design mode for the page:** Enables the end user to drag and drop elements around the page. The end user can use this capability to change the order in which items appear in a zone or to move items from one zone to another.
- **Modify the Web Parts settings:** Enables the end user to customize aspects of the Web Parts, such as their appearance and behavior. It also allows the end user to modify any custom settings that developers apply to the Web Part.

- ❑ **Connect Web Parts on the page:** Enables the end user to make a connection between one or more Web Parts on the page. For example, when an end user working in a financial services application enters a stock symbol into an example Web Part, he can use a connection to another Web Part to see a stock chart change or news appear based on that particular stock symbol. All of this is based on the variable defined in the first Web Part.

Building on Listing 17-2, Listing 17-3 adds a DropDownList control to the table's header. This drop-down list provides a list of available modes the user can employ to change how the page is displayed. Again, the mode of the page determines the actions the user can initiate directly on the page (this is demonstrated later in this chapter).

Listing 17-3: Adding a list of modes to the page

VB

```
<%@ Page Language="VB"%>
<%@ Register Src="DailyLinks.ascx" TagName="DailyLinks" TagPrefix="uc1" %>

<script runat="server">
    Protected Sub DropDownList1_SelectedIndexChanged(ByVal sender As Object, _
       ByVal e As System.EventArgs)

      Dim wpDisplayMode As WebParts.WebPartDisplayMode = _
      Webpartmanager1.SupportedDisplayModes(DropDownList1.SelectedValue.ToString())
      Webpartmanager1.DisplayMode = wpDisplayMode
    End Sub

    Protected Sub Page_Init(ByVal sender As Object, ByVal e As System.EventArgs)
      For Each wpMode As WebPartDisplayMode In _
         Webpartmanager1.SupportedDisplayModes

            Dim modeName As String = wpMode.Name
            Dim dd_ListItem As ListItem = New ListItem(modeName, modeName)
            DropDownList1.Items.Add(dd_ListItem)
      Next
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Web Parts Example</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:WebPartManager ID="Webpartmanager1" Runat="server">
        </asp:WebPartManager>
        <table cellpadding="5" border="1">
            <tr>
                <td colspan="2">
                    <h1>Bill Evjen's Web Page</h1>
                    <asp:WebPartZone ID="WebPartZone1" Runat="server"
                     LayoutOrientation="Horizontal">
                        <ZoneTemplate>
```

Continued

```
                        <asp:Label ID="Label1" Runat="server" Text="Label"
                         Title="Welcome to my web page!">
                         Welcome to the page!
                        </asp:Label>
                    </ZoneTemplate>
                </asp:WebPartZone>
            </td>
            <td valign="top">
                Select mode:
                <asp:DropDownList ID="DropDownList1" runat="server"
                 AutoPostBack="True"
                 OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
                </asp:DropDownList>
            </td>
        </tr>
        <tr valign="top">
            <td>
                <asp:WebPartZone ID="WebPartZone2" Runat="server">
                    <ZoneTemplate>
                        <asp:Image ID="Image1" Runat="server"
                         ImageUrl="~/Images/Tuija.jpg" Width="150px"
                         Title="Tuija at the Museum">
                        </asp:Image>
                        <uc1:DailyLinks ID="DailyLinks1" runat="server"
                         Title="Daily Links">
                        </uc1:DailyLinks>
                    </ZoneTemplate>
                </asp:WebPartZone>
            </td>
            <td>
                <asp:WebPartZone ID="WebPartZone3" Runat="server">
                    <ZoneTemplate>
                        <asp:Calendar ID="Calendar1" Runat="server"
                         Title="Calendar">
                        </asp:Calendar>
                    </ZoneTemplate>
                </asp:WebPartZone>
            </td>
            <td><!-- Blank for now -->
            </td>
        </tr>
    </table>
</form>
</body>
</html>
```

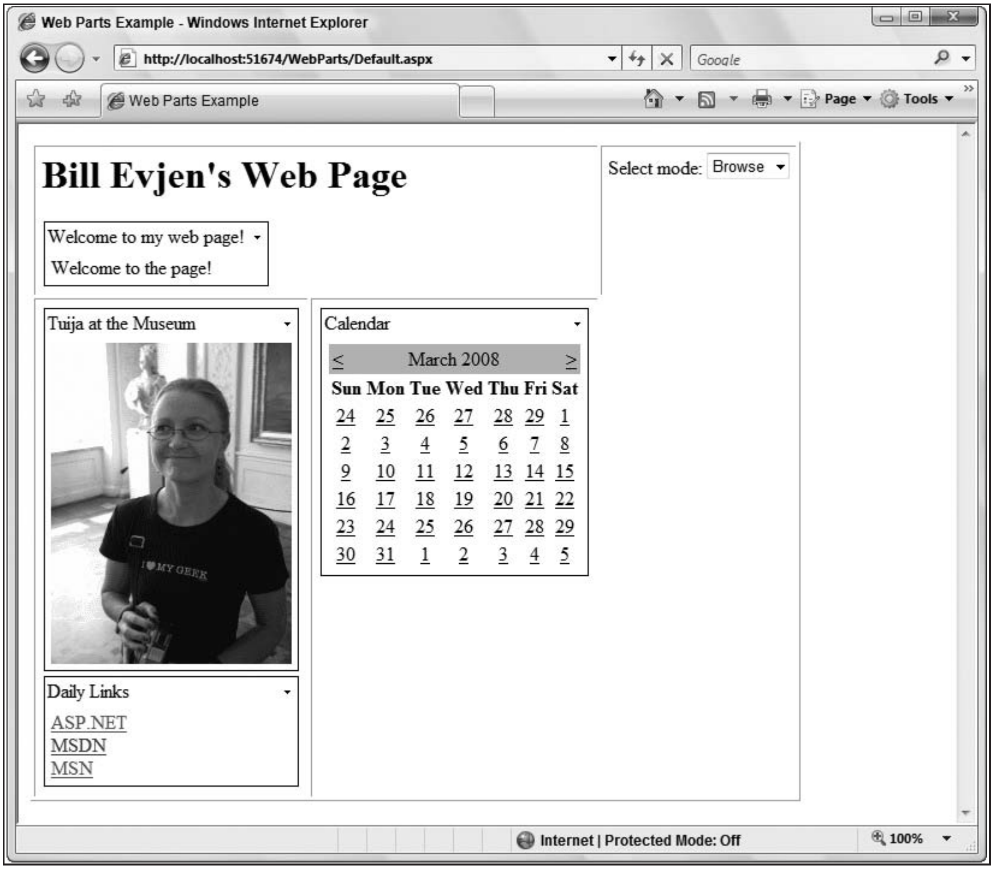
C#

```
<%@ Page Language="C#"%>
<%@ Register Src="DailyLinks.ascx" TagName="DailyLinks" TagPrefix="uc1" %>

<script runat="server">
    protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
    {
      WebParts.WebPartDisplayMode wpDisplayMode =
```

Continued

```
     Webpartmanager1.SupportedDisplayModes[DropDownList1.SelectedValue.ToString()];
      Webpartmanager1.DisplayMode = wpDisplayMode;
    }

    protected void Page_Init(object sender, EventArgs e)
    {
      foreach (WebPartDisplayMode wpMode in
         Webpartmanager1.SupportedDisplayModes)
      {
            string modeName = wpMode.Name;
            ListItem dd_ListItem = new ListItem(modeName, modeName);
            DropDownList1.Items.Add(dd_ListItem);
      }
    }
</script>
```

This adds a drop-down list to the top of the table, as shown in Figure 17-7. This drop-down list will allow the end user to switch between the Browse and Design modes.


Web Parts Example - Windows Internet Explorer
http://localhost:51674/WebParts/Default.aspx
Google
Web Parts Example
Page
Tools
Internet | Protected Mode: Off
100%

Bill Evjen's Web Page

Select mode: Browse

Welcome to my web page!

Welcome to the page!

Tuija at the Museum

Calendar

<			March 2008			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
24	25	26	27	28	29	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Daily Links

ASP.NET
MSDN
MSN

Figure 17-7

When the end user clicks the link, a drop-down window of options appears, as shown in Figure 17-8.

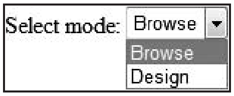


Figure 17-8

Using the `Page_Init` event, the drop-down list is populated with a list of the available page modes that are accessible at this particular time. In this case, it is Browse and Design. The Browse mode is the default mode used when the page is first created. The Design mode causes the ASP.NET page to show the WebPartZone sections. In this mode, the user can drag and drop controls from one section to another with relative ease. Again, the positioning of the elements contained in the page is remembered from one application visit to the next.

It is important to note that the Design mode is only able to work in Internet Explorer browsers.

The DropDownList control is populated by iterating through a list of available `WebPartDisplayMode` objects contained in the `SupportedDisplayModes` property (which is of type `WebPartDisplayModeCollection`). These modes are available through the `WebPartManager1` control, which was placed on the page and is in charge of managing the modes and change of modes of the page. These `WebPartDisplayMode` objects are then used to populate the DropDownList control.

When the end user selects one of the available modes displayed in the DropDownList control, using the `AutoPostBack` feature of the control, the page is then changed to the selected mode. This is done using the first creating an instance of a `WebPartDisplayMode` object and populating it with the value of the mode selected from the drop-down list. Then, using this `WebPartDisplayMode` object, the `DisplayMode` property of the `WebPartManager` object is assigned with this retrieved value.

The next section covers an important addition to the Portal Framework — the capability to add Web Parts dynamically to a page.

Adding Web Parts to a Page

The next step is to rework the example so that the end user has a built-in way to add Web Parts to the page through the use of the Portal Framework. The ASP.NET Portal Framework enables an end user to add Web Parts, but you must also provide the end user with a list of items he can add. To do this, simply add a Catalog Zone to the last table cell in the bottom of the table, as illustrated in the partial code example in Listing 17-4.

Listing 17-4: Adding a Catalog Zone

```
<tr valign="top">
   <td>
      <asp:WebPartZone ID="WebPartZone2" runat="server">
         <ZoneTemplate>
            <asp:Image ID="Image1" runat="server"
             ImageUrl="~/Images/Tuija.jpg" Width="150px"
             Title="Tuija at the Museum">
```

Continued

```
              </asp:Image>
              <uc1:DailyLinks ID="DailyLinks1" runat="server"
               Title="Daily Links">
              </uc1:DailyLinks>
           </ZoneTemplate>
        </asp:WebPartZone>
     </td>
     <td>
        <asp:WebPartZone ID="WebPartZone3" runat="server">
           <ZoneTemplate>
              <asp:Calendar ID="Calendar1" runat="server"
               Title="Calendar">
              </asp:Calendar>
           </ZoneTemplate>
        </asp:WebPartZone>
     </td>
     <td>
        <asp:CatalogZone ID="Catalogzone1" runat="server">
           <ZoneTemplate>
              <asp:PageCatalogPart ID="Pagecatalogpart1" runat="server"/>
           </ZoneTemplate>
        </asp:CatalogZone>
     </td>
  </tr>
```

Once a Catalog Zone section is present on the page, the page is enabled for the Catalog mode. You need to create a Catalog Zone section by using the `<asp:CatalogZone>` control. This is similar to creating a Web Part Zone, but the Catalog Zone is specifically designed to allow for categorization of the items that can be placed on the page. Notice that Catalog mode does not appear as an option in the drop-down list of available modes until a CatalogZone control is placed on the page. If no CatalogZone control is present on the page, this option is not displayed.

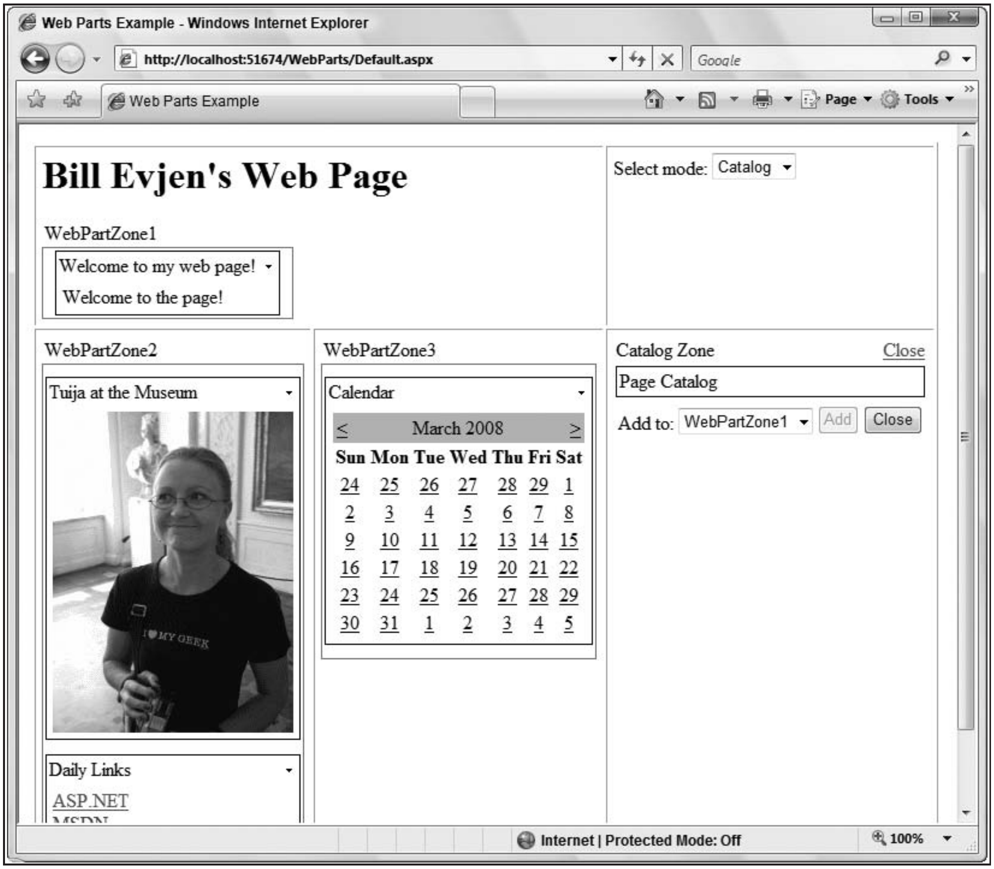
After the Catalog Zone is in place, the next step is to create a `<ZoneTemplate>` section within the Catalog Zone because this is also a templated control. Inside the `<ZoneTemplate>` element is a single control — the PageCatalogPart control. If you run the page after adding the PageCatalogPart control and change the mode to Catalog, you will see the results shown in Figure 17-9.

To get some items to appear in the list (since none do at present), delete one or more items (any items contained on the page when viewing the page in the browser) from the page's default view and enter the Catalog mode by selecting Catalog from the drop-down list of modes.

At this point, you can see the deleted Web Parts in the Catalog Zone. The PageCatalogPart control contains a title and check box list of items that can be selected. The PageCatalogPart control also includes a drop-down list of all the available Web Part Zones on the page. From here, you can place the selected Web Parts into one of the Web Part Zones available from this list. After you select the Web Parts and the appropriate zone in which you want to place the item, you click the Add button and the items appear in the specified locations.

Moving Web Parts

Not only can the end user change the order in which Web Parts appear in a zone, he can also move Web Parts from one zone to another. By adding the capability to enter the Design mode through the drop-down list that you created earlier, you have already provided the end user with this capability. He simply enters the Design mode, and this allows for this type of movement.


Web Parts Example - Windows Internet Explorer
http://localhost:51674/WebParts/Default.aspx
Google
Web Parts Example
Page
Tools
Internet | Protected Mode: Off
100%

Bill Evjen's Web Page

Select mode: Catalog

WebPartZone1

Welcome to my web page!

Welcome to the page!

WebPartZone2

Tuija at the Museum

WebPartZone3

Calendar

<			March 2008			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
24	25	26	27	28	29	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Catalog Zone Close

Page Catalog

Add to: WebPartZone1 Add Close

Daily Links

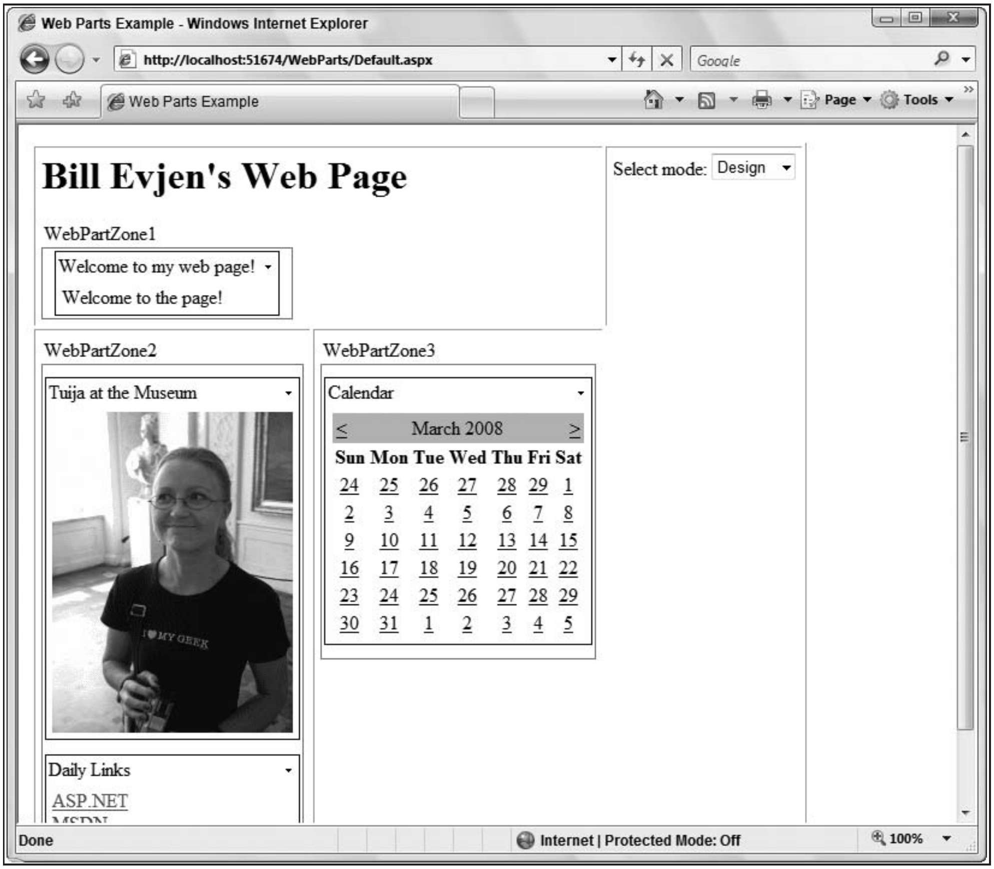
ASP.NET

MSDN

Figure 17-9

The Design option in the drop-down list changes the page so that the user can see the zones defined on the page, as illustrated in Figure 17-10.

From this figure, you can see the three zones (`WebPartZone1`, `WebPartZone2`, and `WebPartZone3`). At this point, the end user can select one of the Web Parts contained in one of these zones and either change its order in the zone or move it to an entirely different zone on the page. To grab one of the Web Parts, the user simply clicks and holds the left mouse button on the title of the Web Part. When done correctly, the crosshair, which appears when the end user hovers over the Web Part's title, turns into an arrow. This means that the user has grabbed hold of the Web Part and can drag it to another part of the page. While the user drags the Web Part around the page, a visual representation of the item appears (see Figure 17-11). In this state, the Web Part is a bit transparent and its location in the state of the page is defined with a blue line (the darker line shown at the top of `WebPartZone3`). Releasing the left mouse button drops the Web Part at the blue line's location.


Web Parts Example - Windows Internet Explorer
http://localhost:51674/WebParts/Default.aspx
Google
Web Parts Example
Page
Tools
Bill Evjen's Web Page
WebPartZone1
WebPartZone2
WebPartZone3
Tuija at the Museum
Calendar
Daily Links
ASP.NET
Done
Internet | Protected Mode: Off
100%

Select mode: Design

Welcome to my web page!

Welcome to the page!

Sun	Mon	Tue	Wed	Thu	Fri	Sat
24	25	26	27	28	29	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Figure 17-10

After the end user places all the items where he wants them, the locations of the items on the page are saved for later use.

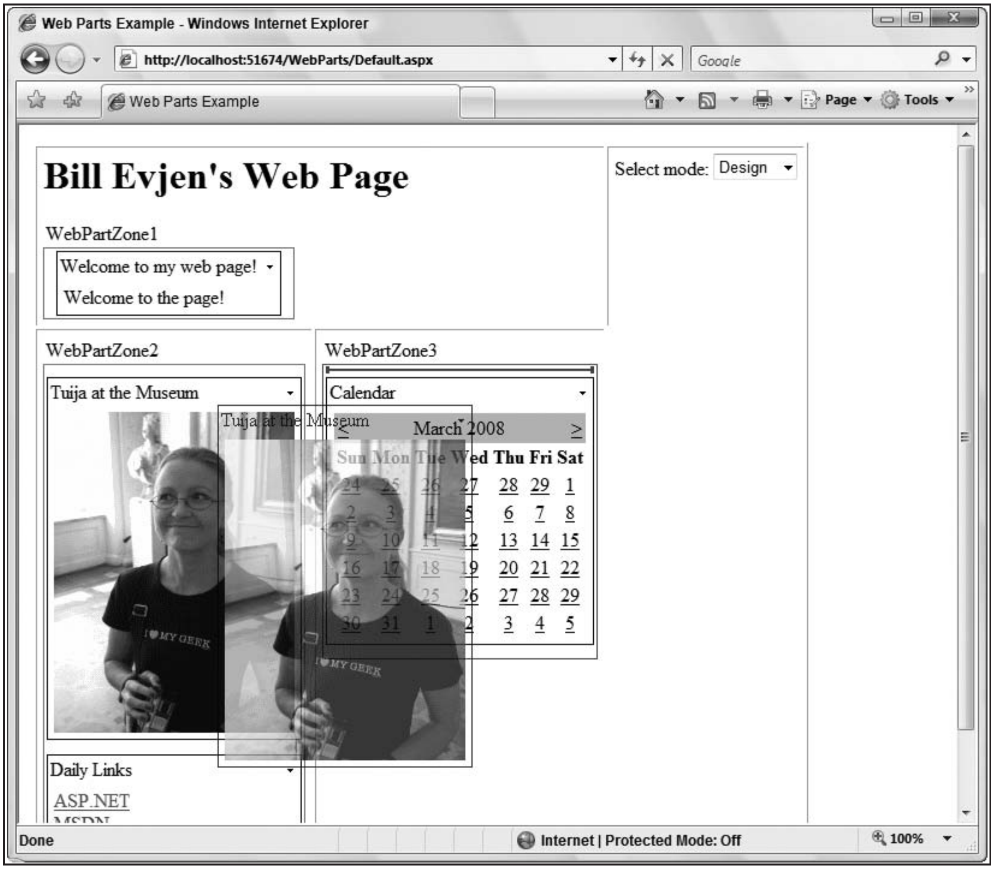
When he reopens the browser, everything is then drawn in the last state in which he left the page. This is done on a per-user basis, so any other users browsing to the same page see either their own modified results or the default view if it is a first visit to the page.

The user can then leave the Design view by opening the list of options from the drop-down list of modes and selecting Browse.

Another way to move Web Parts is to enter the Catalog mode of the page (which is now one of the options in the drop-down list due to the addition of the Catalog Zone section). The Catalog mode enables you to add deleted items to the page, but it also allows you to modify the location of the items on the page by providing the same drag-and-drop capability as the Design mode.

Modifying the Web Part Settings

Another option in the list of modes that can be added to the drop-down list is to allow your end users to edit the actual Web Parts themselves to a degree. This is done through the available Edit mode, and this


Web Parts Example - Windows Internet Explorer
http://localhost:51674/WebParts/Default.aspx
Google
Web Parts Example
Page
Tools
Done
Internet | Protected Mode: Off
100%

Bill Evjen's Web Page

Select mode: Design

WebPartZone1

Welcome to my web page!

Welcome to the page!

WebPartZone2

Tuija at the Museum

Daily Links

ASP.NET

WebPartZone3

Calendar

March 2008

Figure 17-11

enables the end user to modify settings determining appearance, behavior, and layout for a particular Web Part on the page.

To make this functionality work, you must add an Editor Zone to the page just as you add the Catalog Zone. This is illustrated in Listing 17-5. You place this bit of new code within the same table directly below the Catalog Zone declaration.

Listing 17-5: Adding an Editor Zone to the page

```
<td>
   <asp:CatalogZone ID="Catalogzone1" runat="server">
      <ZoneTemplate>
         <asp:PageCatalogPart ID="Pagecatalogpart1" runat="server"/>
      </ZoneTemplate>
   </asp:CatalogZone>
   <asp:EditorZone ID="Editorzone1" runat="server">
      <ZoneTemplate>
         <asp:AppearanceEditorPart ID="Appearanceeditorpart1" runat="server" />
         <asp:BehaviorEditorPart ID="Behavioreditorpart1" runat="server" />
         <asp:LayoutEditorPart ID="Layouteditorpart1" runat="server" />
```

Continued

```
            <asp:PropertyGridEditorPart ID="PropertyGridEditorPart1" runat="server" />
        </ZoneTemplate>
    </asp:EditorZone>
</td>
```

Just like the `<asp:CatalogZone>`, the `<asp:EditorZone>` control is a templated control that requires a `<ZoneTemplate>` section. Within this section, you can place controls that allow for the modification of the appearance, behavior, and layout of the selected Web Part. These controls include `<asp:AppearanceEditorPart>`, `<asp:BehaviorEditorPart>`, `<asp:LayoutEditorPart>`, and `<asp:PropertyGridEditorPart>`.

When you run this new section of code and select Edit from the drop-down list of modes, the arrow that is next to the Web Part title from each of the Web Parts on the page will show an Edit option, as illustrated in Figure 17-12.

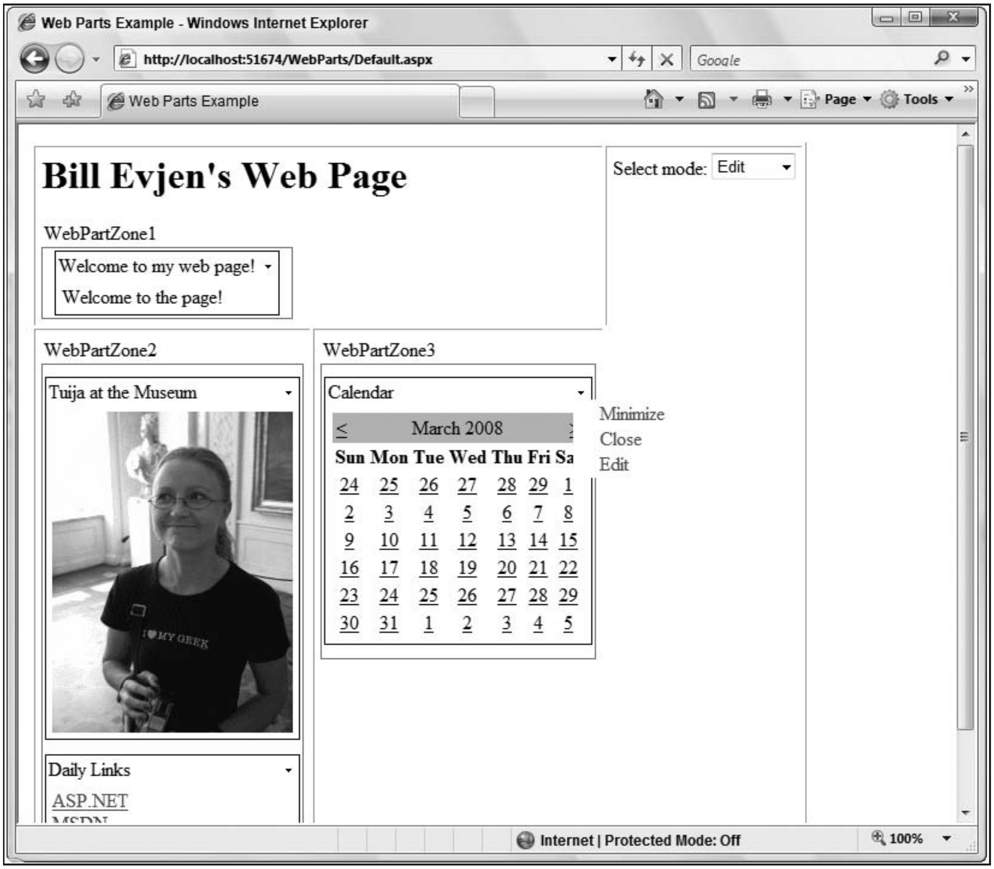
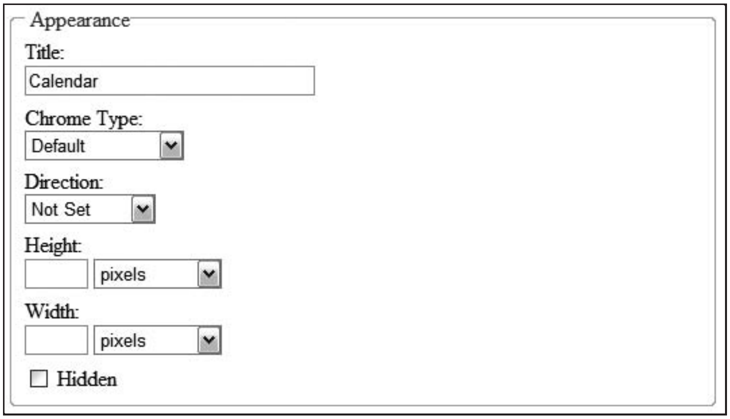

Web Parts Example - Windows Internet Explorer
http://localhost:51674/WebParts/Default.aspx
Google
Web Parts Example
Page
Tools
Bill Evjen's Web Page
Select mode: Edit
WebPartZone1
Welcome to my web page!
Welcome to the page!
WebPartZone2
Tuija at the Museum
Daily Links
ASP.NET
WebPartZone3
Calendar
March 2008
Sun Mon Tue Wed Thu Fri Sa
Minimize
Close
Edit
Internet | Protected Mode: Off
100%

Figure 17-12

After you select the Edit option from this list of three options, the right column of the table shows the various editing sections for this particular Web Part.

The Appearance section enables the end user to change the Web Part's details, including the title, how the title appears, and other appearance-related items such as the item's height and width. The Appearance section is shown in Figure 17-13.



Appearance
Title: Calendar
Chrome Type: Default
Direction: Not Set
Height: pixels
Width: pixels
☐ Hidden

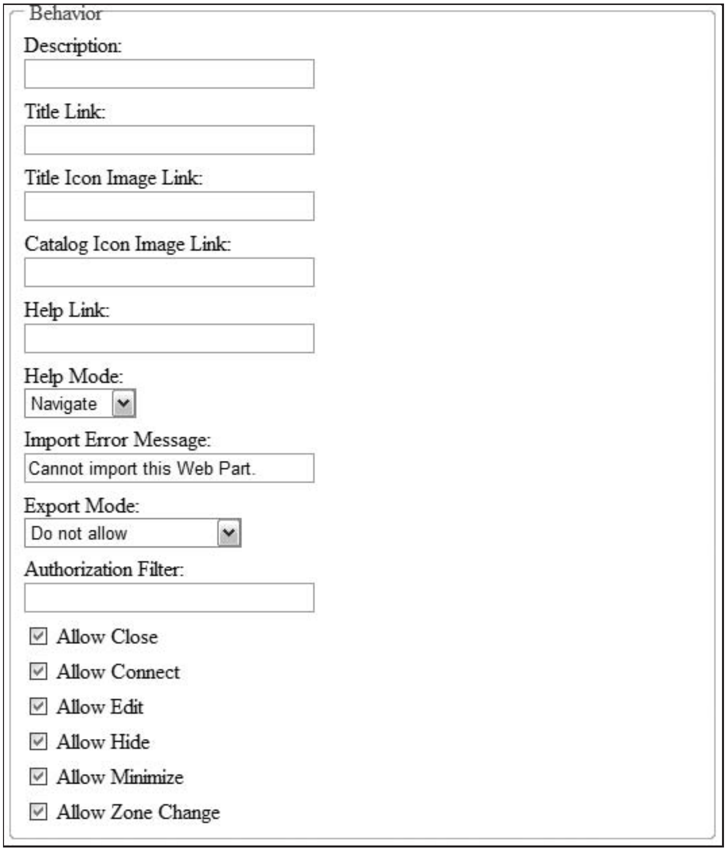
Figure 17-13

The Behavior section (shown in Figure 17-14) enables the end user to select whether the Web Part can be closed, minimized, or exported. This section allows you to change behavior items for either yourself only (a single user) or for everyone in the system (a shared view of the Web Part). Using the shared view, the Behavior section is generally used to allow site editors (or admins) to change the dynamics of how end users can modify Web Parts. General viewers of the page most likely will not see this section.

To get the Behavior section to appear, you first need to make the changes to the `Web.config` files presented in Listing 17-6.

Listing 17-6: Getting the Behavior section to appear through settings in the Web.config

```
<configuration>
   <system.web>
      <webParts>
         <personalization>
            <authorization>
               <allow users="*" verbs="enterSharedScope" />
            </authorization>
         </personalization>
      </webParts>
   </system.web>
</configuration>
```



Behavior

Description:

Title Link:

Title Icon Image Link:

Catalog Icon Image Link:

Help Link:

Help Mode: Navigate

Import Error Message: Cannot import this Web Part.

Export Mode: Do not allow

Authorization Filter:

☑ Allow Close
☑ Allow Connect
☑ Allow Edit
☑ Allow Hide
☑ Allow Minimize
☑ Allow Zone Change

Figure 17-14

After the `Web.config` file is in place, the next step is to add a bit of code to your `Page_Load` event, as shown in Listing 17-7.

Listing 17-7: Adding some code to allow the Behavior section to appear

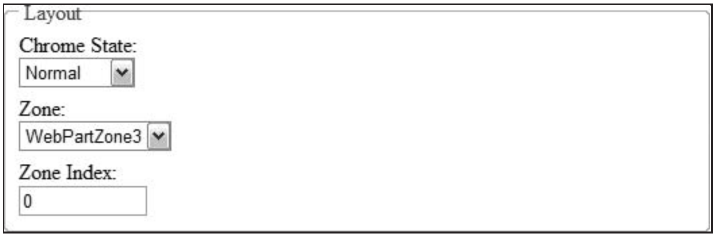
VB

```
If Webpartmanager1.Personalization.Scope = PersonalizationScope.User _
   AndAlso Webpartmanager1.Personalization.CanEnterSharedScope Then

      Webpartmanager1.Personalization.ToggleScope()
End If
```

C#

```
if (Webpartmanager1.Personalization.Scope == PersonalizationScope.User
   && Webpartmanager1.Personalization.CanEnterSharedScope)
{
     Webpartmanager1.Personalization.ToggleScope();
}
```

The Layout section (shown in Figure 17-15) enables the end user to change the order in which Web Parts appear in a zone or move Web Parts from one zone to another. This is quite similar to the drag-and-drop capabilities illustrated previously, but this section allows for the same capabilities through the manipulation of simple form elements.


Layout

Chrome State: Normal

Zone: WebPartZone3

Zone Index: 0

Figure 17-15

The PropertyGridEditorPart, although not demonstrated yet, allows end users to modify properties that are defined in your own custom server controls. At the end of this chapter, we will look at building a custom Web Part and using the PropertyGridEditorPart to allow end users to modify one of the publicly exposed properties contained in the control.

After you are satisfied with the appearance and layout of the Web Parts and have made the necessary changes to the control's properties in one of the editor parts, simply click OK or Apply.

Connecting Web Parts

One option you do have is to make a connection between two Web Parts using the `<asp:ConnectionsZone>` control. This control enables you to make property connections between two Web Parts on the same page. For example, within the Weather Web Part built into one of ASP.NET's pre-built applications, you can have a separate Web Part that is simply a text box and a button that allows the end user to input a zip code. This, in turn, modifies the contents in the original Weather Web Part.

Modifying Zones

One aspect of the Portal Framework that merits special attention is the capability to modify zones on the page. These zones allow for a high degree of modification — not only in the look-and-feel of the items placed in the zone, but also in terms of the behaviors of zones and the items contained in the zones as well. Following are some examples of what you can do to modify zones.

Turning Off the Capability for Modifications in a Zone

As you have seen, giving end users the capability to move Web Parts around the page is quite easy, whether within a zone or among entirely different zones. When working with the Portal Framework and multiple zones on a page, you do not always want to allow the end user to freely change the items that appear in every zone. You want the items placed in some zones to be left alone. Listing 17-8 shows an example of this.

Listing 17-8: Turning off the zone modification capability

```
<asp:WebPartZone ID="WebPartZone1" runat="server"
 LayoutOrientation="Horizontal" AllowLayoutChange="false">
   <ZoneTemplate>
      <asp:Label ID="Label1" runat="server" Text="Label"
       Title="Welcome to my web page!">
       Welcome to the page!
      </asp:Label>
   </ZoneTemplate>
</asp:WebPartZone>
```

In this example, the first Web Part Zone, `WebPartZone1`, uses the `AllowLayoutChange` attribute with a value of `False`, which turns off the end user's capability to modify this particular Web Part Zone. When you run this page and go to the design mode, notice that you cannot drag and drop any of the Web Parts from the other zones into `WebPartZone1`. Neither can you grab hold of the Label Web Part contained in `WebPartZone1`. No capability exists to minimize and close the Web Parts contained in this zone. It allows absolutely no modifications to the zone's layout.

You may notice another interesting change when you are working in the page catalog mode with the `AllowLayoutChange` attribute set to `False`. After you select items to add to the page through the page catalog, `WebPartZone1` does not appear in the drop-down list of places where you can publish the Web Parts (see Figure 17-16). From this figure, you can see that only `WebPartZone2` and `WebPartZone3` appear and allow modifications.

Adding Controls through Other Means

Earlier in this chapter, you examined how to use the `<asp:PageCatalogPart>` control to restore controls to a page after they had been deleted. Although the `<asp:PageCatalogPart>` is ideal for this, you might also want to allow the end user to add Web Parts that are not on the page by default. You may want to enable the end user to add more than one of any particular Web Part to a page. For these situations, you work with the `<asp:DeclarativeCatalogPart>` control.

Listing 17-9 shows an example of using this type of catalog system in place of the `<asp:PageCatalogPart>` control.

Listing 17-9: Using the DeclarativeCatalogPart control

```
<asp:CatalogZone ID="Catalogzone1" Runat="server">
   <ZoneTemplate>
      <asp:DeclarativeCatalogPart ID="Declarativecatalogpart1" Runat="server">
         <WebPartsTemplate>
            <uc1:CompanyContactInfo ID="CompanyContact" Runat="Server"
             Title="Company Contact Info" />
            <uc1:PhotoAlbum ID="PhotoAlbum" Runat="Server" Title="Photo Album" />
            <uc1:Customers ID="Customers" Runat="Server" Title="Customers" />
            <uc1:Locations ID="Locations" Runat="Server" Title="Locations" />
         </WebPartsTemplate>
      </asp:DeclarativeCatalogPart>
   </ZoneTemplate>
</asp:CatalogZone>
```

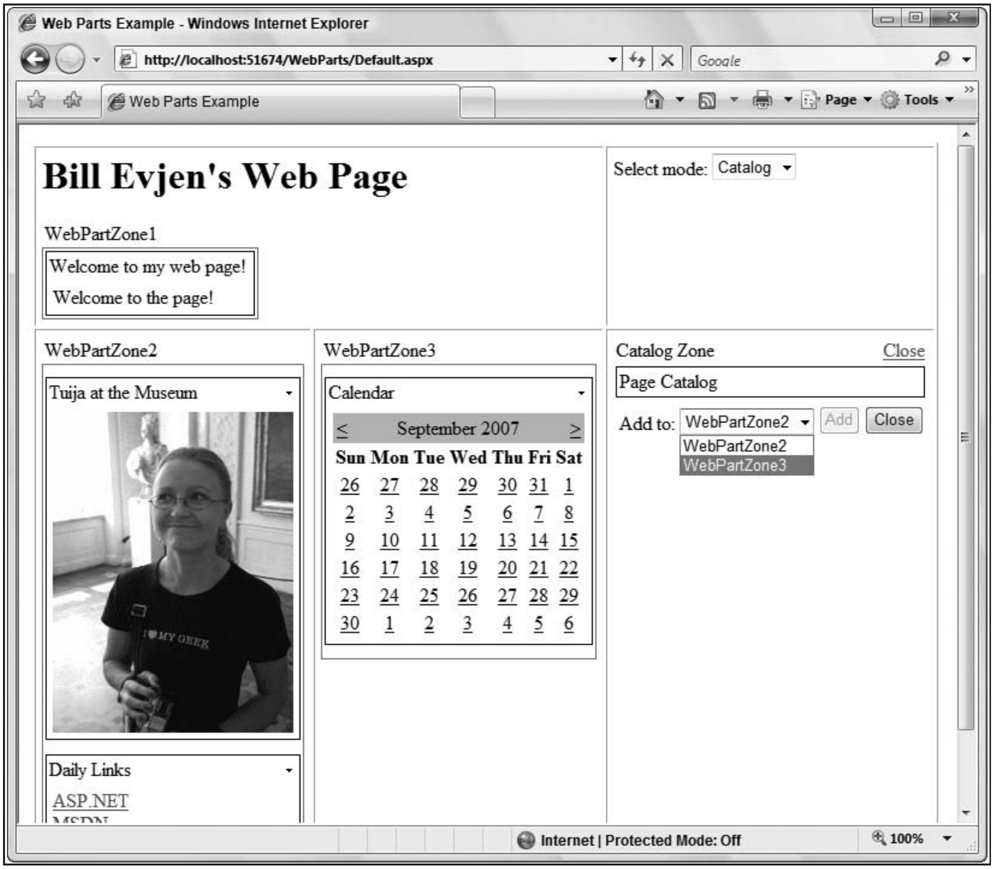
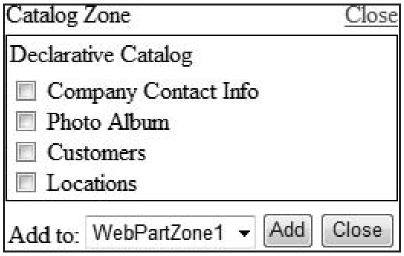

Web Parts Example - Windows Internet Explorer
http://localhost:51674/WebParts/Default.aspx
Google
Web Parts Example
Page
Tools
Bill Evjen's Web Page
Select mode: Catalog
WebPartZone1
Welcome to my web page!
Welcome to the page!
WebPartZone2
Tuija at the Museum
Daily Links
ASP.NET
MSDN
WebPartZone3
Calendar
September 2007
Sun Mon Tue Wed Thu Fri Sat
Catalog Zone
Close
Page Catalog
Add to: WebPartZone2
Add
Close
WebPartZone2
WebPartZone3
Internet | Protected Mode: Off
100%

Figure 17-16

Instead of using the `<asp:PageCatalogPart>` control, this catalog uses the `<asp:DeclarativeCatalogPart>` control. This templated control needs a `<WebPartsTemplate>` section where you can place all the controls you want available as options for the end user. The controls appear in the check box list in the same order in which you declare them in the `<WebPartsTemplate>` section. Figure 17-17 shows how the catalog looks in the Design view in Visual Studio 2008.



Catalog Zone Close

Declarative Catalog

☐ Company Contact Info
☐ Photo Album
☐ Customers
☐ Locations

Add to: WebPartZone1 Add Close

Figure 17-17

This catalog lets you select items from the list of Web Parts and assign the location of the zone in which they will be placed. After they are placed, notice that the option to add these Web Parts has not disappeared as it did with the earlier PageCatalogPart control. In fact, you can add as many of these items to the page as you deem necessary — even if it is to the same zone within the Portal Framework.

Using the DeclarativeCatalogPart control is not always a completely ideal solution. When the end user closes one of the Web Parts that initially appears on the page, he may not see that control listed in the DeclarativeCatalogPart control's list of elements. You must explicitly specify it should appear when you write the code for the DeclarativeCatalogPart control. In fact, the end user cannot re-add these deleted items. Using both the PageCatalogPart control and the DeclarativeCatalogPart control simultaneously is sometimes the best solution. The great thing about this framework is that it allows you to do that. The Portal Framework melds both controls into a cohesive control that not only enables you to add controls that are not on the page by default, but it also lets you add previously deleted default controls. Listing 17-10 shows an example of this.

Listing 17-10: Combining both catalog types

```
<asp:CatalogZone ID="Catalogzone1" Runat="server">
   <ZoneTemplate>
      <asp:PageCatalogPart ID="Pagecatalogpart1" Runat="server" />
      <asp:DeclarativeCatalogPart ID="Declarativecatalogpart1" Runat="server">
         <WebPartsTemplate>
            <uc1:CompanyContactInfo ID="CompanyContact" Runat="Server"
             Title="Company Contact Info" />
            <uc1:PhotoAlbum ID="PhotoAlbum" Runat="Server" Title="Photo Album" />
            <uc1:Customers ID="Customers" Runat="Server" Title="Customers" />
            <uc1:Locations ID="Locations" Runat="Server" Title="Locations" />
         </WebPartsTemplate>
      </asp:DeclarativeCatalogPart>
   </ZoneTemplate>
</asp:CatalogZone>
```

In this example, both the PageCatalogPart control and the DeclarativeCatalogPart control are contained within the `<ZoneTemplate>` section. When this page is run, you see the results shown in Figure 17-18.

You can see that each catalog is defined within the Catalog Zone. Figure 17-18 shows the PageCatalogPart control's collection of Web Parts (defined as Page Catalog). Also, note that a link to the Declarative Catalog is provided for that particular list of items. Note that the order in which the catalogs appear in the `<ZoneTemplate>` section is the order in which the links appear in the Catalog Zone.

Web Part Verbs

Web Part verbs declare the actions of the items (such as Minimize and Close) that appear in the title. These verbs are basically links that initiate an action for a particular Web Part. The available list of Web Part verbs includes the following:

- ❑ `<CloseVerb>`
- ❑ `<ConnectVerb>>`
- ❑ `<DeleteVerb>`
- ❑ `<EditVerb>`

- ❑ `<ExportVerb>`
- ❑ `<HelpVerb>`
- ❑ `<MinimizeVerb>`
- ❑ `<RestoreVerb>`

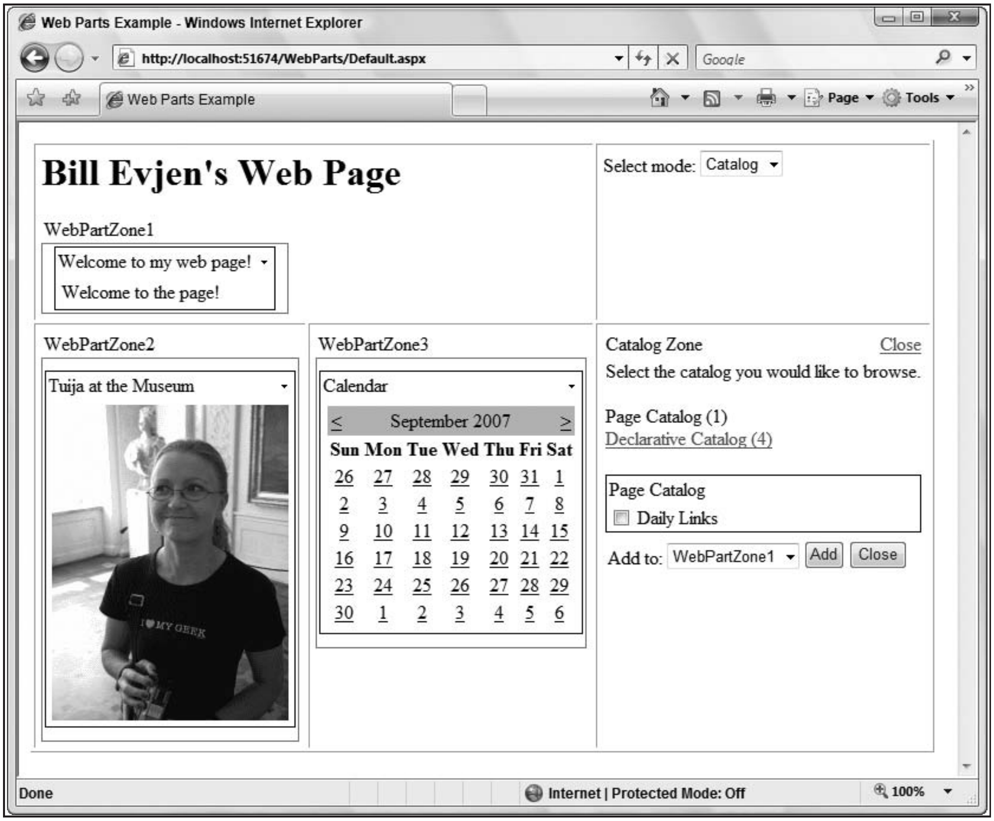

Web Parts Example - Windows Internet Explorer
http://localhost:51674/WebParts/Default.aspx
Google
Web Parts Example
Page
Tools
Bill Evjen's Web Page
Select mode: Catalog
WebPartZone1
Welcome to my web page!
Welcome to the page!
WebPartZone2
WebPartZone3
Tuija at the Museum
Calendar
September 2007
Sun Mon Tue Wed Thu Fri Sat
26 27 28 29 30 31 1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 1 2 3 4 5 6
Catalog Zone
Close
Select the catalog you would like to browse.
Page Catalog (1)
Declarative Catalog (4)
Page Catalog
☐ Daily Links
Add to: WebPartZone1
Add
Close
Done
Internet | Protected Mode: Off
100%

Figure 17-18

The `<asp:WebPartZone>` control allows you to control these verbs by nesting the appropriate verb elements within the `<asp:WebPartZone>` element itself. After these are in place, you can manipulate how these items appear in all the Web Parts that appear in the chosen Web Part Zone.

For example, look at graying out the default Close link included with a Web Part. This is illustrated in Listing 17-11.

Listing 17-11: Graying out the Close link in a Web Part

```
<asp:WebPartZone ID="WebPartZone3" Runat="server">
     <CloseVerb Enabled="False" />
     <ZoneTemplate>
        <asp:Calendar ID="Calendar1" Runat="server">
```

Continued

```
            </asp:Calendar>
        </ZoneTemplate>
    </asp:WebPartZone>
```

In this example, you can see that you simply need to set the `Enabled` attribute of the `<CloseVerb>` element to `False` in order to gray out the Close link in any of the generated Web Parts included in this Web Part Zone. If you construct the Web Part Zone in this manner, you achieve the results shown in Figure 17-19.

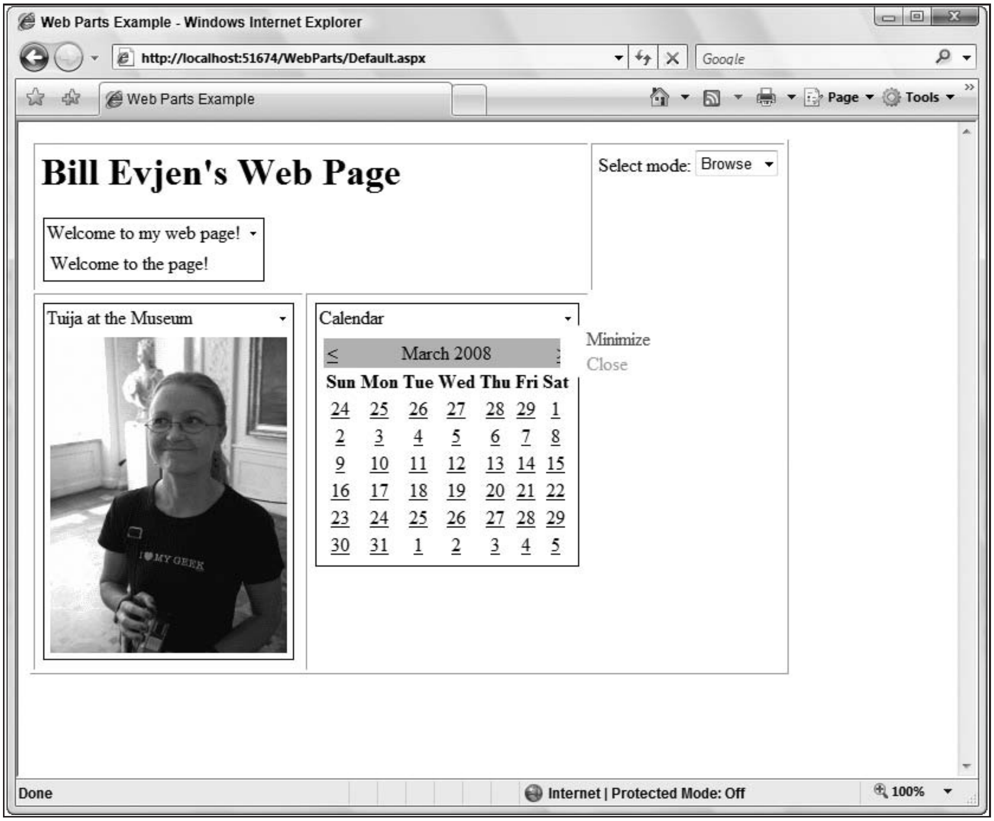

Web Parts Example - Windows Internet Explorer
http://localhost:51674/WebParts/Default.aspx
Google
Web Parts Example
Page
Tools
Bill Evjen's Web Page
Select mode: Browse
Welcome to my web page!
Welcome to the page!
Tuija at the Museum
Calendar
Minimize
Close
March 2008
Sun Mon Tue Wed Thu Fri Sat
24 25 26 27 28 29 1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31 1 2 3 4 5
Done
Internet | Protected Mode: Off
100%

Figure 17-19

If you do not want to gray out the Close link (or any other verb link contained within the Web Part), you must instead use the `Visible` attribute of the appropriate verb (see Listing 17-12).

Listing 17-12: Removing the Close link in a Web Part

```
    <asp:WebPartZone ID="WebPartZone3" Runat="server">
        <CloseVerb Visible="False" />
```

Continued

```
      <ZoneTemplate>
         <asp:Calendar ID="Calendar1" Runat="server">
         </asp:Calendar>
      </ZoneTemplate>
</asp:WebPartZone>
```

Using the `Visible` attribute produces the screen shown in Figure 17-20.

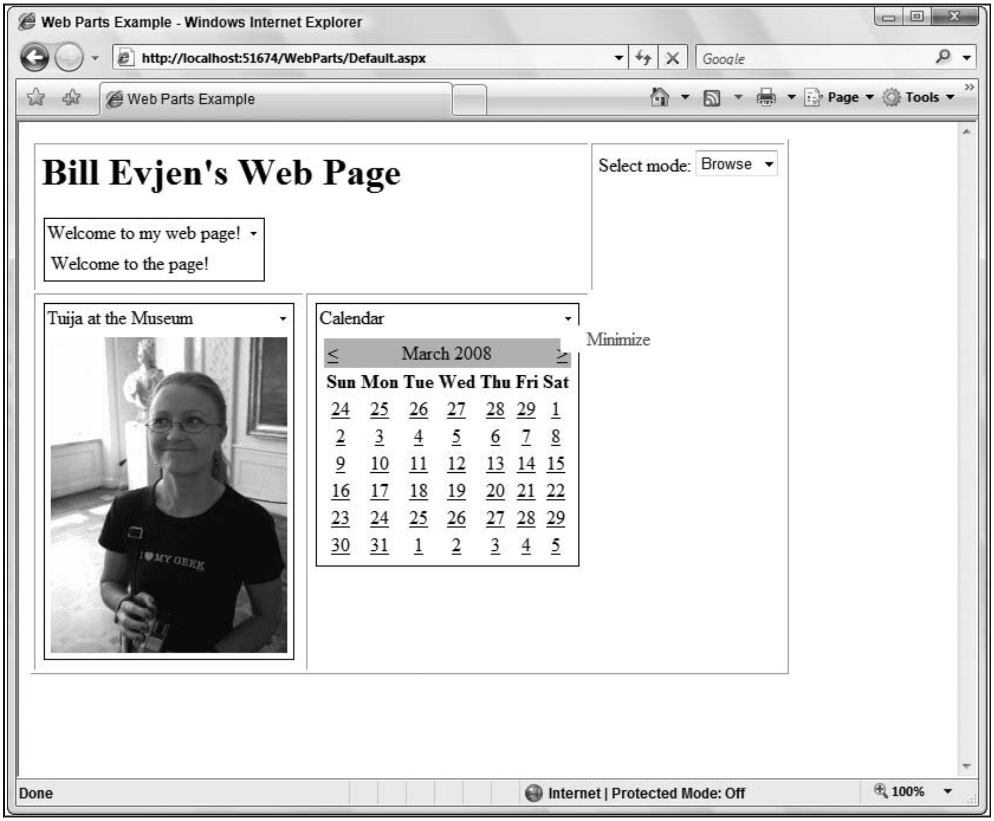

Web Parts Example - Windows Internet Explorer
http://localhost:51674/WebParts/Default.aspx
Google
Web Parts Example
Page
Tools
Bill Evjen's Web Page
Select mode: Browse
Welcome to my web page!
Welcome to the page!
Tuija at the Museum
Calendar
Minimize
< March 2008 >
Sun Mon Tue Wed Thu Fri Sat
24 25 26 27 28 29 1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31 1 2 3 4 5
Done
Internet | Protected Mode: Off
100%

Figure 17-20

Verb elements provide another exciting feature: They give you the capability to use images that would appear next to the text of an item. Using images with the text makes the Web Parts appear more like the overall Windows environment. For instance, you can change the contents of `WebPartZone3` again so that it now uses images with the text for the Close and Minimize links. This is illustrated in Listing 17-13.

Listing 17-13: Using images for the Web Part verbs

```
<asp:WebPartZone ID="WebPartZone3" Runat="server">
      <CloseVerb ImageUrl="Images/CloseVerb.gif" />
```

Continued

```
        <MinimizeVerb ImageUrl="Images/MinimizeVerb.gif" />
        <ZoneTemplate>
           <asp:Calendar ID="Calendar1" Runat="server">
           </asp:Calendar>
        </ZoneTemplate>
  </asp:WebPartZone>
```

To point to an image for the verb, use the `ImageUrl` attribute. This produces something similar to Figure 17-21, depending on the images you use.

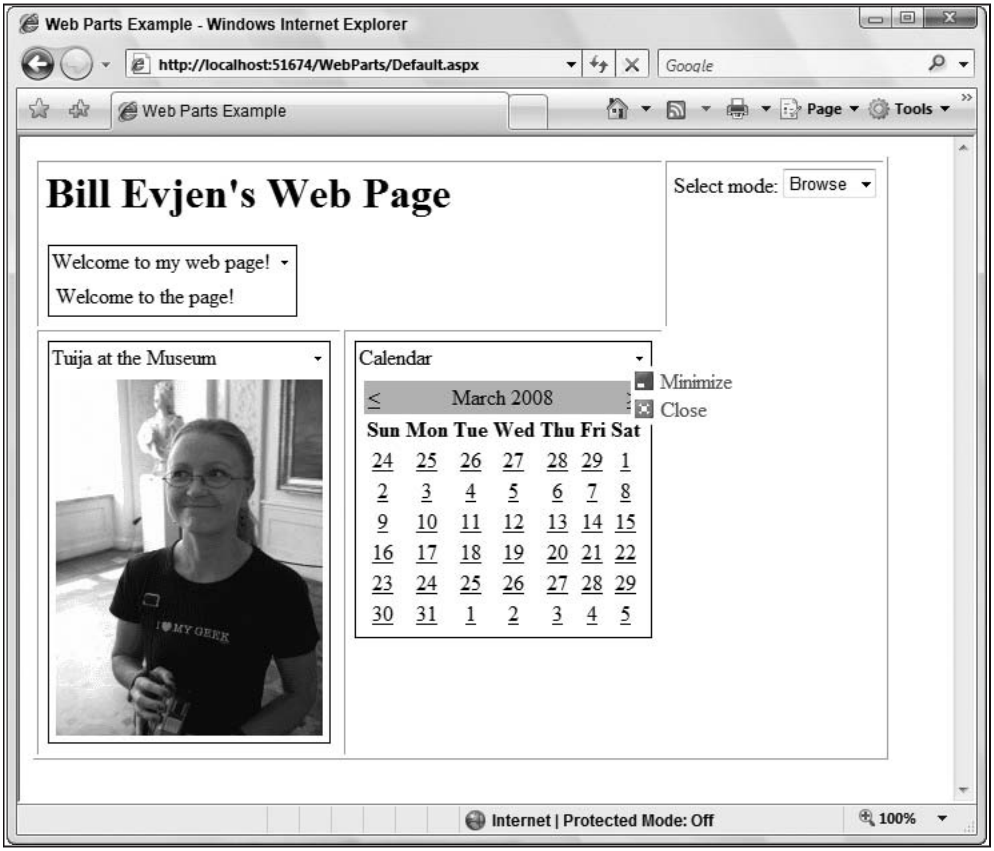

Web Parts Example - Windows Internet Explorer
http://localhost:51674/WebParts/Default.aspx
Google
Web Parts Example
Page
Tools
Bill Evjen's Web Page
Select mode:
Browse
Welcome to my web page!
Welcome to the page!
Tuija at the Museum
Calendar
Minimize
Close
March 2008
Sun Mon Tue Wed Thu Fri Sat
Internet | Protected Mode: Off
100%

Figure 17-21

This chapter, thus far, has concentrated on creating completely customizable portal applications in a declarative manner using the capabilities provided by the ASP.NET Portal Framework. As with most aspects of ASP.NET, however, not only can you work with appearance and functionality in a declarative fashion, but you can also create the same constructs through server-side code.

Working with Classes in the Portal Framework

The Portal Framework provides three main classes for dealing with the underlying framework presented in this chapter: `WebPartManager`, `WebPartZone`, and `WebPart`.

The `WebPartManager` class allows you to perform multiple operations in your server-side code. The following table shows a partial listing of some of the properties that this class provides.

WebPartManager Class Properties	Description
`Connections`	Provides a collection of all the connections between Web Parts contained on the page.
`DisplayMode`	Allows you to change the page's display mode. Possible choices include `CatalogDisplayMode`, `ConnectDisplayMode`, `DesignDisplayMode`, `EditDisplayMode`, and `BrowseDisplayMode`.
`SelectedWebPart`	Allows you to perform multiple operations on the selected Web Part.
`WebParts`	Provides a collection of all the Web Parts contained on the page.
`Zones`	Provides a collection of all the Web Part Zones contained on the page.

Beyond the properties of the `WebPartManager` class, you also have an extensive list of available methods at your disposal. The following table outlines some of the available methods of the `WebPartManager` class.

WebPartManager Class Methods	Description
`AddWebPart`	Allows you to dynamically add new Web Parts to a particular zone on the page.
`ConnectWebParts`	Allows you to connect two Web Parts together via a common property or value.
`DeleteWebPart`	Allows you to dynamically delete new Web Parts from a particular zone on the page.
`DisconnectWebParts`	Allows you to delete a connection between two Web Parts.
`MoveWebPart`	Allows you to move a Web Part from one zone to another, or allows you to change the index order in which Web Parts appear in a particular zone.

Whereas the `WebPartManager` class allows you to manipulate the location, addition, and deletion of Web Parts that appear in the page as a whole, the `WebPartZone` class allows you to modify a single Web Part Zone on the page. The following table provides a list of some properties available to the `WebPartZone` class.

WebPartZone Class Properties	Description
`AllowLayoutChange`	Takes a `Boolean` value and either enables or disables the Web Part Zone's capability to accept or allow any changes in the Web Parts it contains.
`BackColor`, `BackImageUrl`, `BorderColor`, `BorderStyle`, `BorderWidth`	Enable you to modify the Web Part Zone's general appearance.
`CloseVerb`	References the Close verb for a particular Web Part Zone from which you can then manipulate the verb's `Description`, `Enabled`, `ImageUrl`, `Text`, and `Visible` properties.
`ConnectVerb`	References a Web Part Zone's Connect verb from which you can then manipulate the verb's `Description`, `Enabled`, `ImageUrl`, `Text`, and `Visible` properties.
`DragHighlightColor`	Takes a `System.Color` value that sets the color of the Web Part Zone's border if focused when the moving of Web Parts is in operation. This also changes the color of the line that appears in the Web Part Zone specifying where to drop the Web Part.
`EditVerb`	References a Web Part Zone's Edit verb from which you can then manipulate the verb's `Description`, `Enabled`, `ImageUrl`, `Text`, and `Visible` properties.
`EmptyZoneText`	Sets the text that is shown in the zone if a Web Part is not set in the zone.
`HeaderText`	Sets header text.
`Height`	Sets the height of the Web Part Zone.
`HelpVerb`	References a Web Part Zone's Help verb from which you can then manipulate the verb's `Description`, `Enabled`, `ImageUrl`, `Text`, and `Visible` properties.
`MenuLabelStyle`, `MenuLabelText`	Enable you to modify the drop-down menu that appears when end users edit a Web Part. These properties let you apply an image, alter the text, or change the style of the menu.
`MinimizeVerb`	References a Web Part Zone's Minimize verb from which you can then manipulate the verb's `Description`, `Enabled`, `ImageUrl`, `Text`, and `Visible` properties.
`LayoutOrientation`	Enables you to change the Web Part Zone's orientation from horizontal to vertical or vice versa.
`RestoreVerb`	References a Web Part Zone's Restore verb, from which you can then manipulate the verb's `Description`, `Enabled`, `ImageUrl`, `Text`, and `Visible` properties.

WebPartZone Class Properties	Description
`VerbButtonType`	Enables you to change the button style. Choices include `ButtonType.Button`, `ButtonType.Image`, or `ButtonType.Link`.
`WebParts`	Provides a collection of all the Web Parts contained within the zone.
`Width`	Sets the width of the Web Part Zone.

You have a plethora of options to manipulate the look-and-feel of the Web Part Zone and the items contained therein.

The final class is the `WebPart` class. This class enables you to manipulate specific Web Parts located on the page. The following table details some of the properties available in the `WebPart` class.

WebPart Class Properties	Description
`AllowClose`	Takes a `Boolean` value that specifies whether the Web Part can be closed and removed from the page.
`AllowEdit`	Takes a `Boolean` value that specifies whether the end user can edit the Web Part.
`AllowHide`	Takes a `Boolean` value that specifies whether the end user can hide the Web Part within the Web Part Zone. If the control is hidden, it is still in the zone, but invisible.
`AllowMinimize`	Takes a `Boolean` value that specifies whether the end user can collapse the Web Part.
`AllowZoneChange`	Takes a `Boolean` value that specifies whether the end user can move the Web Part from one zone to another.
`BackColor`, `BackImageUrl`, `BorderColor`, `BorderStyle`, `BorderWidth`	Enable you to modify the Web Part's general appearance.
`ChromeState`	Specifies whether the Web Part chrome is in a normal state or is minimized.
`ChromeType`	Specifies the chrome type that the Web Part uses. Available options include `BorderOnly`, `Default`, `None`, `TitleAndBorder`, and `TitleOnly`.
`Direction`	Specifies the direction of the text or items placed within the Web Part. Available options include `LeftToRight`, `NotSet`, and `RightToLeft`. This property is ideal for dealing with Web Parts that contain Asian text that is read from right to left.

WebPart Class Properties	**Description**
`HelpMode`	Specifies how the help items display when the end user clicks the Help verb. Available options include `Modal`, `Modeless`, and `Navigate`. `Modal` displays the help items within a modal window if the end user's browser supports modal windows. If not, a pop-up window displays. `Modeless` means that a pop-up window displays for every user. `Navigate` redirects the user to the appropriate help page (specified by the `HelpUrl` property) when he clicks on the Help verb.
`HelpUrl`	Used when the `HelpMode` is set to `Navigate`. Takes a `String` value that specifies the location of the page the end user is redirected to when he clicks on the Help verb.
`ScrollBars`	Applies scroll bars to the Web Part. Available values include `Auto`, `Both`, `Horizontal`, `None`, and `Vertical`.
`Title`	Specifies the text for the Web Part's title. Text appears in the title bar section.
`TitleIconImageUrl`	Enables you to apply an icon to appear next to the title by specifying to the icon image's location as a `String` value of the property.
`TitleUrl`	Specifies the location to direct the end user when the Web Part's title Web Part is clicked. When set, the title is converted to a link; when not set, the title appears as regular text.
`Zone`	Allows you to refer to the zone in which the Web Part is located.

Creating Custom Web Parts

When adding items to a page that utilizes the Portal Framework, you add the pre-existing ASP.NET Web server controls, user controls, or custom controls. In addition to these items, you can also build and incorporate custom Web Parts. Using the `WebParts` class, you can create your own custom Web Parts. Although similar to ASP.NET custom server control development, the creation of custom Web Parts adds some additional capabilities. Creating a class that inherits from the `WebPart` class instead of the `Control` class enables your control to use the new personalization features and to work with the larger Portal Framework, thereby allowing for the control to be closed, maximized, minimized, and more.

To create a custom Web Part control, the first step is to create a project in Visual Studio 2008. From Visual Studio, choose File ➪ New Project. This pops open the New Project dialog. From this dialog, select ASP.NET Server Control. Name the project `MyStateListBox` and click OK to create the project. You are presented with a class that contains the basic framework for a typical ASP.NET server control. Ignore this framework; you are going to change it so that your class creates a custom Web Parts control instead of an ASP.NET custom server control. Listing 17-14 details the creation of a custom Web Part control.

Listing 17-14: Creating a custom Web Part control

VB

```
Imports System
Imports System.Web
Imports System.Web.UI.WebControls
Imports System.Web.UI.WebControls.WebParts

Namespace Wrox

    Public Class StateListBox
        Inherits WebPart

        Private _LabelStartText As String = " Enter State Name: "
        Dim StateInput As New TextBox
        Dim StateContents As New ListBox

        Public Sub New()
            Me.AllowClose = False
        End Sub

        <Personalizable(), WebBrowsable()> _
        Public Property LabelStartText() As String
            Get
                Return _LabelStartText
            End Get
            Set(ByVal value As String)
                _LabelStartText = value
            End Set
        End Property

        Protected Overrides Sub CreateChildControls()
            Controls.Clear()

            Dim InstructionText As New Label
            InstructionText.BackColor = Drawing.Color.LightGray
            InstructionText.Font.Name = "Verdana"
            InstructionText.Font.Size = 10
            InstructionText.Font.Bold = True
            InstructionText.Text = LabelStartText
            Me.Controls.Add(InstructionText)

            Dim LineBreak As New Literal
            LineBreak.Text = "<br />"
            Me.Controls.Add(LineBreak)

            Me.Controls.Add(StateInput)

            Dim InputButton As New Button
            InputButton.Text = "Input State"
            AddHandler InputButton.Click, AddressOf Me.Button1_Click
            Me.Controls.Add(InputButton)
```

Continued

```
            Dim Spacer As New Literal
            Spacer.Text = "<p>"
            Me.Controls.Add(Spacer)

            Me.Controls.Add(StateContents)

            ChildControlsCreated = True
        End Sub

        Public Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
            StateContents.Items.Add(StateInput.Text)
            StateInput.Text = String.Empty
            StateInput.Focus()
        End Sub

    End Class

End Namespace
```

C#

```
using System;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

namespace Wrox
{
    public class StateListBox : WebPart
    {
        private String _LabelStartText = " Enter State Name: ";
        readonly TextBox StateInput = new TextBox();
        readonly ListBox StateContents = new ListBox();

        public StateListBox()
        {
            AllowClose = false;
        }

        [Personalizable, WebBrowsable]
        public String LabelStartText
        {
            get { return _LabelStartText; }
            set { _LabelStartText = value; }
        }

        protected override void CreateChildControls()
        {
            Controls.Clear();

            Label InstructionText = new Label();
            InstructionText.BackColor = System.Drawing.Color.LightGray;
            InstructionText.Font.Name = "Verdana";
            InstructionText.Font.Size = 10;
            InstructionText.Font.Bold = true;
            InstructionText.Text = LabelStartText;
```

Continued

```
            Controls.Add(InstructionText);

            Literal LineBreak = new Literal();
            LineBreak.Text = "<br />";
            Controls.Add(LineBreak);

            Controls.Add(StateInput);

            Button InputButton = new Button();
            InputButton.Text = "Input State";
            InputButton.Click += this.Button1_Click;
            Controls.Add(InputButton);

            Literal Spacer = new Literal();
            Spacer.Text = "<p>";
            Controls.Add(Spacer);

            Controls.Add(StateContents);

            ChildControlsCreated = true;
        }

        private void Button1_Click(object sender, EventArgs e)
        {
            StateContents.Items.Add(StateInput.Text);
            StateInput.Text = String.Empty;
            StateInput.Focus();
        }
    }
}
```

To review, you first import the `System.Web.UI.WebControls.WebParts` namespace. The important step in the creation of this custom control is to make sure that it inherits from the `WebPart` class instead of the customary `Control` class. As stated earlier, this gives the control access to the advanced functionality of the Portal Framework that a typical custom control would not have.

VB

```
Public Class StateListBox
   Inherits WebPart

End Class
```

C#

```
public class StateListBox : WebPart
{

}
```

After the class structure is in place, a few properties are defined, and the constructor is defined as well. The constructor directly uses some of the capabilities that the `WebPart` class provides. These capabilities would not be available if this custom control has the `Control` class as its base class and is making use of the `WebPart.AllowClose` property.

VB

```
Public Sub New()
   Me.AllowClose = False
End Sub
```

C#

```
public StateListBox()
{
   AllowClose = false;
}
```

This constructor creates a control that explicitly sets the control's `AllowClose` property to `False` — meaning that the Web Part will not have a Close link associated with it when generated in the page. Because of the use of the `WebPart` class instead of the `Control` class, you will find, in addition to the `AllowClose` property, other `WebPart` class properties such as `AllowEdit`, `AllowHide`, `AllowMinimize`, `AllowZoneChange`, and more.

In the example shown in Listing 17-14, you see a custom-defined property: `LabelStartText`. This property allows the developer to change the instruction text displayed at the top of the control. The big difference with this custom property is that it is preceded by the `Personalizable` and the `WebBrowsable` attributes.

The `Personalizable` attribute enables the property for personalization, whereas the `WebBrowsable` attribute specifies whether the property should be displayed in the Properties window in Visual Studio. The `Personalizable` attribute can be defined further using a `PersonalizationScope` enumeration. The only two possible enumerations — `Shared` and `User` — can be defined in the following ways:

VB

```
<Personalizable(PersonalizationScope.Shared), WebBrowsable()> _
Public Property LabelStartText() As String
   Get
      Return _LabelStartText
   End Get
   Set(ByVal value As String)
      _LabelStartText = value
   End Set
End Property
```

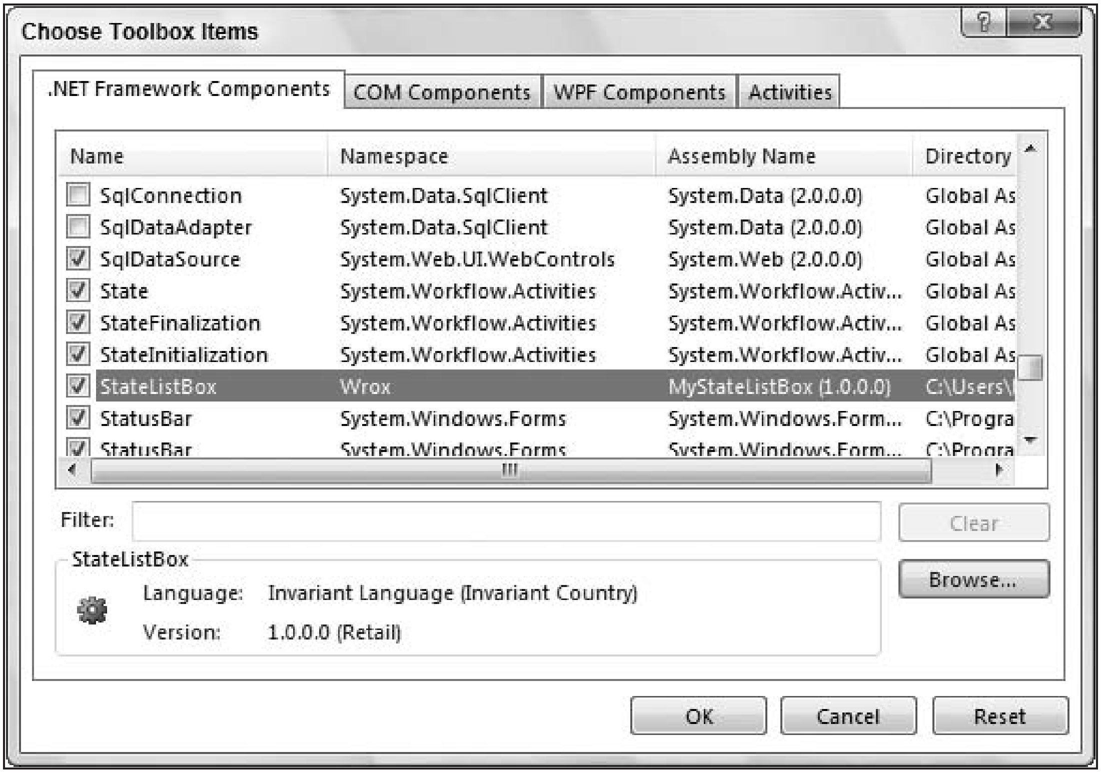
C#

```
[Personalizable(PersonalizationScope.Shared), WebBrowsable]
public String LabelStartText
{
   get { return _LabelStartText; }
   set { _LabelStartText = value; }
}
```

A `PersonalizationScope` of `User` means that any modifications are done on a per-user basis. This is the default setting and means that if a user makes modifications to the property, the changes are seen only by that particular user and not by the other users that browse the page. If the `PersonalizationScope` is set to `Shared`, changes made by one user can be viewed by others requesting the page.

After you have any properties in place, the next step is to define what gets rendered to the page by overriding the `CreateChildControls` method. From the example in Listing 17-14, the `CreateChildControls` method renders Label, Literal, TextBox, Button, and ListBox controls. In addition to defining the properties of some of these controls, a single event is associated with the Button control (`Button1_Click`) that is also defined in this class.

Now that the custom Web Part control is in place, build the project so that a DLL is created. The next step is to open up the ASP.NET Web project where you want to utilize this new control and, from the Visual Studio Toolbox, add the new control. You can quickly accomplish this task by right-clicking in the Toolbox on the tab where you want the new control to be placed. After right-clicking the appropriate tab, select Choose Items. Click the Browse button and point to the new `MyStateListBox.dll` that you just created. After this is done, the StateListBox control is highlighted and checked in the Choose Toolbox Items dialog, as illustrated in Figure 17-22.



Name	Namespace	Assembly Name	Directory
☐ SqlConnection	System.Data.SqlClient	System.Data (2.0.0.0)	Global As
☐ SqlDataAdapter	System.Data.SqlClient	System.Data (2.0.0.0)	Global As
☑ SqlDataSource	System.Web.UI.WebControls	System.Web (2.0.0.0)	Global As
☑ State	System.Workflow.Activities	System.Workflow.Activ...	Global As
☑ StateFinalization	System.Workflow.Activities	System.Workflow.Activ...	Global As
☑ StateInitialization	System.Workflow.Activities	System.Workflow.Activ...	Global As
☑ StateListBox	Wrox	MyStateListBox (1.0.0.0)	C:\Users\
☑ StatusBar	System.Windows.Forms	System.Windows.Form...	C:\Progra
☑ StatusBar	System.Windows.Forms	System.Windows.Form...	C:\Progra

StateListBox
Language: Invariant Language (Invariant Country)
Version: 1.0.0.0 (Retail)

Figure 17-22

Clicking OK adds the control to your Toolbox. Now you are ready to use this new control as a Web Part control. To do this, simply drag and drop the control into one of your Web Part Zone areas. This does a couple of things. First, it registers the control on the page using the Register directive:

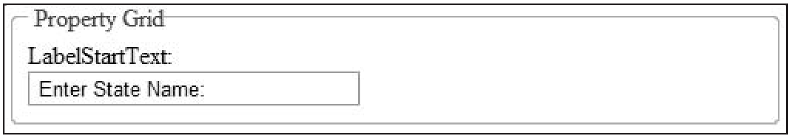
```
<%@ Register TagPrefix="cc1" Namespace="MyStateListBox.Wrox"
    Assembly="MyStateListBox" %>
```

Once registered, the control can be used on the page. If dragged and dropped onto the page's design surface, you get a control in the following construct:

```
<cc1:StateListBox Runat="server" ID="StateListBox1"
 LabelStartText=" Enter State Name: " AllowClose="False" />
```

The two important things to notice with this construct is that the custom property, `LabelStartText`, is present and has the default value in place, and the `AllowClose` attribute is included. The `AllowClose` attribute is present only because earlier you made the control's inherited class `WebPart`and not `Control`. Because `WebPart` was made the inherited class, you have access to these Web Part–specific properties. When the StateListBox control is drawn on the page, you can see that, indeed, it is part of the larger Portal Framework and allows for things such as minimization and editing. End users can use this custom Web Part control as if it were any other type of Web Part control. As you can see, you have a lot of power when you create your own Web Part controls.

And because `LabelStartText` uses the `WebBrowsable` attribute, you can use the PropertyGridEditorPart control to allow end users to edit this directly in the browser. With this in place, as was demonstrated earlier in Listing 17-5, an end user will see the following editing capabilities after switching to the Edit mode (see Figure 17-23).



LabelStartText:

Enter State Name:

Figure 17-23

Connecting Web Parts

In working with Web Parts, you sometimes need to connect them in some fashion. C*onnecting* them means that you must pass a piece of information (an object) from one Web Part to another Web Part on the page.

For instance, you might want to transfer the text value (such as a zip code or a name) that someone enters in a text box to other Web Parts in the page. Another example is a DropDownList control that specifies all the available currencies in the system. If the end user selects from the drop-down list, this drives changes in all the other Web Parts on that page that deal with this currency value selection. When you need to build constructions in this manner, you can use the Web Part connection capabilities defined here, or you might be able to work with other ASP.NET systems available (such as the personalization capabilities provided through the profile system).

When connecting Web Parts, you should be aware of the specific rules on how these Web Parts interact with one another. First off, if you want to make a connection from one Web Part to another, one of the Web Parts must be the *provider*. This provider Web Part is the component that supplies the piece of information required by any other Web Parts. The Web Parts that require this information are the *consumer* Web Parts. A Web Part provider can supply information to one or more consumer Web Parts; however, a consumer Web Part can only connect with a single provider Web Part. You cannot have a consumer Web Part that connects to more than one provider Web Part.

An example of this scenario is illustrated in Figure 17-24.

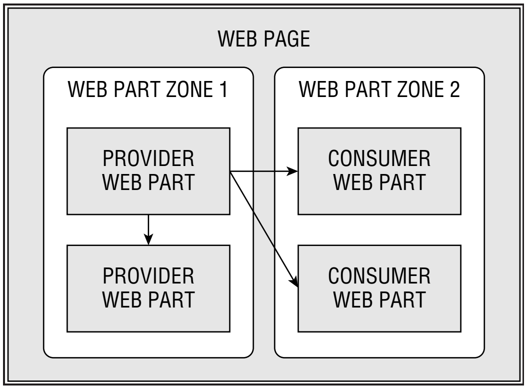


Figure 17-24

From this diagram, you can see that it is possible to use a single provider Web Part with multiple consumer Web Parts regardless of the Web Part Zone area in which the consumer Web Part resides.

When working with provider and consumer Web Parts, be aware that no matter how simple or complicated the Web Part is, you must wrap the Web Part by making it a custom Web Part. You do this to expose the correct item or property from the provider Web Part or to utilize the passed value in the correct manner for the consumer Web Part. This chapter reviews a simple example of connecting Web Parts. Although many steps are required to accomplish this task, you definitely get a lot of value from it as well.

Building the Provider Web Part

In order to build a provider Web Part that can be utilized by any consumer Web Part residing on the page, you first create an interface exposing the item you want to pass from one Web Part to another.

For this example, suppose you want to provide a text box as a Web Part that allows the end user to input a value. This value is then utilized by a calendar control contained within a Web Part in an entirely different Web Part Zone on the page using the Calendar control's `Caption` property.

This interface is demonstrated in Listing 17-15.

Listing 17-15: Building an interface to expose the property used to pass to a Web Part

VB

```
Namespace Wrox.ConnectionManagement
    Public Interface IStringForCalendar
        Property CalendarString() As String
    End Interface
End Namespace
```

Continued

C#

```
namespace Wrox.ConnectionManagement
{
    public interface IStringForCalendar
    {
        string CalendarString { get; set;}
    }
}
```

From this bit of code, you can see that the interface, `IStringForCalendar`, is quite simple. It exposes only a single String property — `CalendarString`. This interface is then utilized by the custom provider Web Part shown next.

The custom provider Web Part is built like any other custom Web Part in the manner demonstrated earlier in this chapter. The only difference is that you also provide some extra details so ASP.NET knows what item you are exposing from the Web Part and how this value is retrieved. This custom provider Web Part is illustrated in Listing 17-16.

Listing 17-16: Building a custom provider Web Part

VB

```
Imports Microsoft.VisualBasic
Imports System.Web.UI.WebControls
Imports System.Web.UI.WebControls.WebParts

Namespace Wrox.ConnectionManagement
    Public Class TextBoxChanger
        Inherits WebPart
        Implements IStringForCalendar

        Private myTextBox As TextBox
        Private _calendarString As String = String.Empty

        <Personalizable()> _
        Public Property CalendarString() As String Implements _
          Wrox.ConnectionManagement.IStringForCalendar.CalendarString

            Get
                Return _calendarString
            End Get
            Set(ByVal value As String)
                _calendarString = value
            End Set

        End Property

        <ConnectionProvider("Provider for String From TextBox", _
            "TextBoxStringProvider")> _
        Public Function TextBoxStringProvider() As IStringForCalendar
            Return Me
        End Function
```

Continued

```
        Protected Overrides Sub CreateChildControls()
            Controls.Clear()
            myTextBox = New TextBox()
            Me.Controls.Add(myTextBox)
            Dim myButton As Button = New Button()
            myButton.Text = "Change Calendar Caption"
            AddHandler myButton.Click, AddressOf Me.myButton_Click
            Me.Controls.Add(myButton)
        End Sub

        Private Sub myButton_Click(ByVal sender As Object, ByVal e As EventArgs)
            If myTextBox.Text <> String.Empty Then
                CalendarString = myTextBox.Text
                myTextBox.Text = String.Empty
            End If
        End Sub
    End Class
End Namespace
```

C#

```
using System;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

namespace Wrox.ConnectionManagement
{
    public class TextBoxChanger : WebPart, IStringForCalendar
    {
        private TextBox myTextBox;
        private string _calendarString = String.Empty;

        [Personalizable]
        public string CalendarString
        {
            get { return _calendarString; }
            set { _calendarString = value; }
        }

        [ConnectionProvider("Provider for String From TextBox",
            "TextBoxStringProvider")]
        public IStringForCalendar TextBoxStringProvider()
        {
            return this;
        }

        protected override void CreateChildControls()
        {
            Controls.Clear();
            myTextBox = new TextBox();
            Controls.Add(myTextBox);
            Button myButton = new Button();
            myButton.Text = "Change Calendar Caption";
            myButton.Click += this.myButton_Click;
```

Continued

```
                Controls.Add(myButton);
            }

            private void myButton_Click(object sender, EventArgs e)
            {
                if (myTextBox.Text != String.Empty)
                {
                    CalendarString = myTextBox.Text;
                    myTextBox.Text = String.Empty;
                }
            }
        }
    }
```

Not only does this Web Part inherit the `WebPart` class, this provider Web Part implements the interface, `IStringForCalendar`, which was created earlier in Listing 17-15. From `IStringForCalendar`, the single `String` property is utilized and given the `Personalizable()` attribute. The important bit from the code presented in Listing 17-16 is the `TextBoxStringProvider()` method. This method returns a type of `IStringForCalendar` and is defined even further using the `ConnectionProvider` attribute. This attribute enables you to give a friendly name to the provider as well a programmatic name that can be used within the consumer custom Web Part that will be built shortly.

Using the `CreateChildControls()` method, you layout the design of the Web Part. In this case, a text box and button are the only controls that make up this Web Part. In addition to simply laying out the controls on the design surface, you use this method to assign any specific control events — such as a button-click event by assigning handlers to the controls that require them.

Finally, from within the button-click event of this Web Part (`myButton_Click`), the exposed property of `IStringForCalendar` is assigned a value. Everything is now in place for a consumer Web Part. The construction of this second Web Part is demonstrated next.

Building the Consumer Web Part

After you have a provider Web Part in place on your page, you can utilize the object it supplies for any number of consumer Web Parts on the same page. In this example, however, you are interested in using the `String` value coming from the TextBox control in the provider Web Part. The consumer Web Part takes this `String` and assigns it as the value of the Calendar control's `Caption` property. The construction of the consumer Web Part is illustrated in Listing 17-17.

Listing 17-17: The consumer Web Part

VB

```
Imports Microsoft.VisualBasic
Imports System.Web.UI.WebControls
Imports System.Web.UI.WebControls.WebParts

Namespace Wrox.ConnectionManagement
    Public Class ModifyableCalendar
        Inherits WebPart
```

Continued

```
        Private _myProvider As IStringForCalendar
        Private _stringTitle As String
        Private myCalendar As Calendar = New Calendar()

        <ConnectionConsumer("Calendar Title Consumer", "CalendarTitleConsumer")> _
        Public Sub RetrieveTitle(ByVal Provider As IStringForCalendar)
            _myProvider = Provider
        End Sub

        Protected Overrides Sub OnPreRender(ByVal e As EventArgs)
            EnsureChildControls()

            If Not (Me._myProvider Is Nothing) Then
                _stringTitle = _myProvider.CalendarString.Trim()
                myCalendar.Caption = _stringTitle
            End If
        End Sub

        Protected Overrides Sub CreateChildControls()
            Controls.Clear()
            Me.Controls.Add(myCalendar)
        End Sub
    End Class
End Namespace
```

C#

```
using System;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

namespace Wrox.ConnectionManagement
{
    public class ModifyableCalendar : WebPart
    {
        private IStringForCalendar _myProvider;
        string _stringTitle;
        Calendar myCalendar;

        [ConnectionConsumer("Calendar Title Consumer", "CalendarTitleConsumer")]
        public void RetrieveTitle(IStringForCalendar Provider)
        {
            _myProvider = Provider;
        }

        protected override void OnPreRender(EventArgs e)
        {
            EnsureChildControls();

            if (_myProvider != null)
            {
                _stringTitle = _myProvider.CalendarString.Trim();
                myCalendar.Caption = _stringTitle;
            }
```

Continued

```
        }

        protected override void CreateChildControls()
        {
            Controls.Clear();
            myCalendar = new Calendar();
            Controls.Add(myCalendar);
        }
    }
}
```

This new custom Web Part, `ModifyableCalendar`, is simply a class that inherits from `WebPart` and nothing more. It requires a reference to the interface `IStringForCalendar`. Because `IStringForCalendar` is part of the same namespace, you can provide a simple reference to this interface.

Your consumer Web Part requires a method that is the connection point for a provider Web Part on the same page. In this case, the `RetrieveTitle()` method is constructed using the `ConnectionConsumer` attribute before the method declaration. Like the `ConnectionProvider` attribute that was utilized in the provider Web Part, the `ConnectionConsumer` Web Part enables you to give your Web Part a friendly name and a reference name to use programmatically.

Then, the value retrieved from the provider Web Part is grabbed from within the `PreRender()` method of the Web Part and assigned to the Calendar control before the actual Calendar control is placed on the page from within the `CreateChildControls()` method.

Now that both a provider and a consumer Web Part are available to you, the next step is to get them both on an ASP.NET page and build the mechanics to tie the two Web Parts together. This is demonstrated next.

Connecting Web Parts on an ASP.NET Page

When in the process of connecting Web Parts, remember that you need a provider Web Part and a consumer Web Part. These items are detailed in Listings 17-16 and 17-17, respectively. When working with the process of connecting Web Parts, it is not simply a matter of placing both of these items on the page to get the connections to take place. In addition to this step, you have to wire the Web Parts together.

This wiring of Web Part connections is done through the WebPartManager control that is discussed at the beginning part of this chapter. The ASP.NET page used for this example is detailed in Listing 17-18.

Listing 17-18: The ASP.NET page that connects two Web Part controls

```
<%@ Page Language="VB" %>
<%@ Register Namespace="Wrox.ConnectionManagement"
    TagPrefix="connectionControls" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Connecting Web Parts</title>
</head>
<body>
```

Continued

```
    <form id="form1" runat="server">
    <div>
        <asp:WebPartManager ID="WebPartManager1" runat="server">
            <StaticConnections>
                <asp:WebPartConnection ID="WebPartConnection1"
                 ConsumerID="ModifyableCalendar1"
                 ConsumerConnectionPointID="CalendarTitleConsumer"
                 ProviderID="TextBoxChanger1"
                 ProviderConnectionPointID="TextBoxStringProvider">
                </asp:WebPartConnection>
            </StaticConnections>
        </asp:WebPartManager>
        <table cellpadding="3">
            <tr valign="top">
                <td style="width: 100px">
                    <asp:WebPartZone ID="WebPartZone1" runat="server">
                        <ZoneTemplate>
                            <connectionControls:TextBoxChanger
                             ID="TextBoxChanger1"
                             runat="server" Title="Provider Web Part" />
                        </ZoneTemplate>
                    </asp:WebPartZone>
                </td>
                <td style="width: 100px">
                    <asp:WebPartZone ID="WebPartZone2" runat="server">
                        <ZoneTemplate>
                            <connectionControls:ModifyableCalendar
                             ID="ModifyableCalendar1" runat="server"
                             Title="Consumer Web Part" />
                        </ZoneTemplate>
                    </asp:WebPartZone>
                </td>
            </tr>
        </table>
    </div>
    </form>
</body>
</html>
```

This ASP.NET page that utilizes Web Parts contains a single two-cell table. Each cell in the table contains a single WebPartZone control — `WebPartZone1` and `WebPartZone2`.

Before connecting the Web Parts, the new custom Web Part controls are registered in the ASP.NET page using the `@Register` page directive. This directive simply points to the namespace `Wrox.ConnectionManagement`. This is the namespace used by the interface and the two custom Web Part controls.

Each of the custom Web Parts is placed within its own WebPartZone control. The two Web Part controls are tied together using the WebPartManager control.

```
<asp:WebPartManager ID="WebPartManager1" runat="server">
   <StaticConnections>
      <asp:WebPartConnection ID="WebPartConnection1"
       ConsumerID="ModifyableCalendar1"
```

```
        ConsumerConnectionPointID="CalendarTitleConsumer"
        ProviderID="TextBoxChanger1"
        ProviderConnectionPointID="TextBoxStringProvider">
       </asp:WebPartConnection>
    </StaticConnections>
</asp:WebPartManager>
```

The WebPartManager server control nests the defined connection inside of the `<StaticConnections>` section of the declaration. The definition is actually accomplished using the WebPartConnection server control. This control takes four important attributes required in order to make the necessary connections. The first set of two attributes deals with definitions of the consumer settings. Of these, the `ConsumerID` attribute references the name of the control on the ASP.NET page (through its ID attribute) and the `ConsumerConnectionPointID` references the ID of the object working as the connection point for the consumer. Looking back, you find this is the `RetrieveTitle()` method shown in the following code snippet:

```
<ConnectionConsumer("Calendar Title Consumer", "CalendarTitleConsumer")> _
 Public Sub RetrieveTitle(ByVal Provider As IStringForCalendar)
    _myProvider = Provider
 End Sub
```

The second set of attributes required by the WebPartConnection deals with the provider Web Part. The first attribute of this set is the `ProviderID` attribute that makes reference to the name of the control on the ASP.NET page, which is considered the provider. The second attribute, `ProviderConnectionPointID` is quite similar to the `ConsumerConnectionPointID` attribute, but the `ProviderConnectionPointID` attribute references the ID of the object working as the provider in the connection process.

```
<ConnectionProvider("Provider for String From TextBox", "TextBoxStringProvider")> _
 Public Function TextBoxStringProvider() As IStringForCalendar
    Return Me
 End Function
```

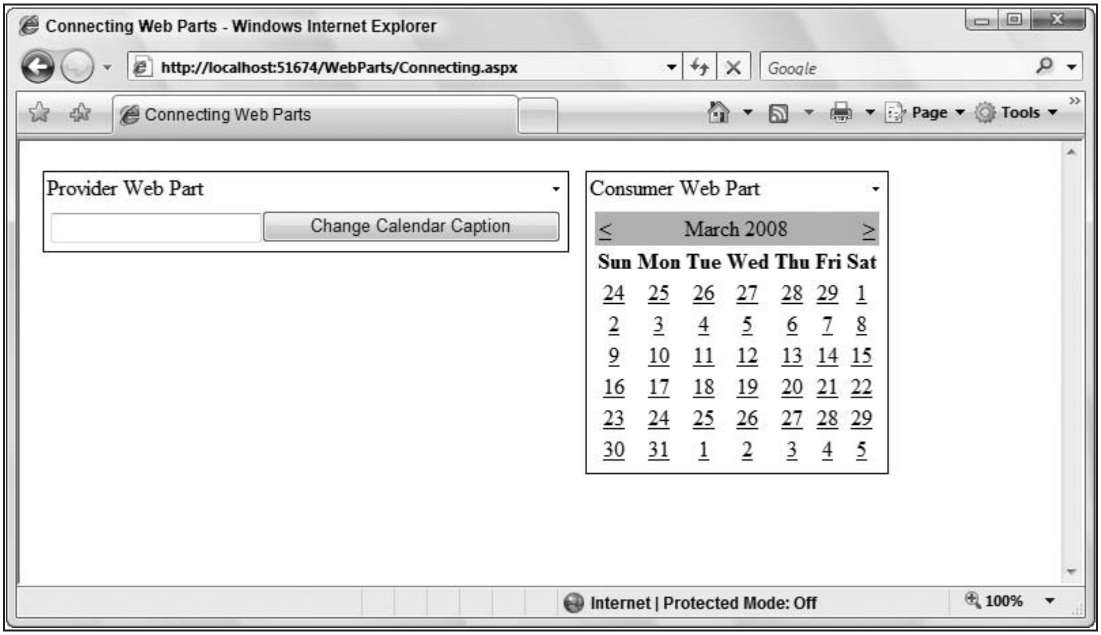
Running this page gives you the results illustrated in Figure 17-25.

If you type any text string in the text box in the provider Web Part on the page and click the button within this control, the Calendar control uses this `String` value as its value for the `Caption` property. This is demonstrated in Figure 17-26.

As you can see from this example, you take a lot of steps to take to make this happen, but the steps aren't too difficult. In this example, a simple `String` object was passed from one Web Part to another. You could, however, use the exact same process to pass more complex objects (even custom objects) or larger items such a `DataSet` object.

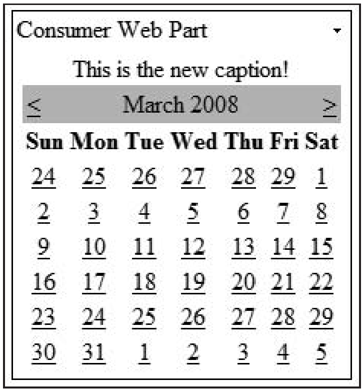
Understanding the Difficulties in Dealing with Master Pages When Connecting Web Parts

You should note one final consideration about dealing with connecting Web Parts on your ASP.NET pages. You might have already realized that this process gets rather difficult when you are working with ASP.NET pages that make use of the master page capability provided by ASP.NET.


Connecting Web Parts - Windows Internet Explorer
http://localhost:51674/WebParts/Connecting.aspx
Google
Connecting Web Parts
Page
Tools
Provider Web Part
Change Calendar Caption
Consumer Web Part
Internet | Protected Mode: Off
100%

<			March 2008			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
24	25	26	27	28	29	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Figure 17-25


Consumer Web Part

This is the new caption!

<			March 2008			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
24	25	26	27	28	29	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Figure 17-26

You are allowed only a single WebPartManager control on an ASP.NET page. Many times, when you are working with master pages, it makes a lot of sense to put this control in the master page itself rather than in the content pages that make use of the master page. If you are taking this approach, it does not make much sense to start using WebPartConnection controls within the master page. You can easily have controls with the same ID on multiple content pages. If you do so, the references made within the WebPartConnection control might not be meant for these other controls. For this reason, you need to make use of the ProxyWebPartManager control.

Suppose you have a master page with a WebPartManger control. In this case, the WebPartManager control can be rather simple, as shown here:

```
<asp:WebPartManager ID="WebPartManager1" runat="server">
</asp:WebPartManager>
```

With this WebPartManager control on your `.master` page, you ensure this single instance manages the Web Parts contained in each and every content page making use of this particular master page.

Next, if a content page making use of this master page is attempting to connect some Web Parts, you must place a ProxyWebPartManger control on the content page itself. This instance of the ProxyWebPartManager is where you define the connections for the Web Parts on this particular content page. This is illustrated in the following code snippet:

```
<asp:ProxyWebPartManager ID="ProxyWebPartManager1" runat="server">
   <StaticConnections>
      <asp:WebPartConnection ID="WebPartConnection1"
       ConsumerID="ModifyableCalendar1"
       ConsumerConnectionPointID="CalendarTitleConsumer"
       ProviderID="TextBoxChanger1"
       ProviderConnectionPointID="TextBoxStringProvider">
      </asp:WebPartConnection>
   </StaticConnections>
</asp:ProxyWebPartManager>
```

Summary

This chapter introduced you to the WebPartManager, WebPartZone, and the WebPart controls. Not only do these controls allow for easy customization of the look-and-feel of either the Web Parts or the zones in which they are located, but also the framework provided can be used to completely modify the behavior of these items.

This chapter also showed you how to create your own custom Web Part controls. Creating your own controls was always one of the benefits provided by ASP.NET, and this benefit has been taken one step further with the capability to now create Web Part controls. Web Part controls enable you to take advantage of some of the more complex features that you do not get with custom ASP.NET server controls.

You may find the Portal Framework to be one of the more exciting features of ASP.NET 3.5; you may like the idea of creating completely modular and customizable Web pages. End users like this feature, and it is quite easy for developers to implement. Just remember that you do not have to implement every feature explained in this chapter; with the framework provided, however, you can choose the functionality that you want.

18

HTML and CSS Design with ASP.NET

When HTML was first introduced by Tim Berners-Lee, it was intended to be a simple way for researchers using the Internet to format and cross-link their research documents. At the time, the Web was still primarily text-based; therefore, the formatting requirements for these documents were fairly basic. HTML needed only a small handful of basic layout concepts such as a title, paragraph, headers, and lists. As the Web was opened up to the general public, graphical browsers were introduced, and as requirements for formatting Web pages continued to expand, newer versions of HTML were introduced. These newer versions expanded the original capabilities of HTML to accommodate the new, rich graphical browser environment, allowing table layouts, richer font styling, images, and frames.

While all of these improvements to HTML were helpful, HTML still proved to be inadequate for allowing developers to create complex, highly stylized Web pages. Therefore, in 1994 a new technology called Cascading Style Sheets was introduced. CSS served as a complementary technology to HTML, giving developers of Web pages the power they needed to finely control the style of their Web pages.

As the Web has matured, CSS has gained popularity as developers realized that it has significant advantages over standard HTML styling capabilities. Unlike HTML, which was originally conceived as primarily a layout mechanism, CSS was conceived from the beginning to provide rich styling capabilities to Web pages. The cascading nature of CSS makes it easy to apply styles with a broad stroke to an entire application, and only where necessary override those styles. CSS makes is easy to externally define Web site style information, allowing for a clear separation of Web page style and structure. CSS also allows developers to greatly reduce the file size of a Web page, which translates into faster page load times and reduced bandwidth consumption.

While the point of this chapter is not to convince you that CSS is the best solution for styling your Web site, it will help you understand how you can leverage these technologies in your ASP.NET-based Web applications. It will start with a brief overview of CSS and how it works with HTML, and then move into creating Web sites in Visual Studio using HTML and CSS. Finally, you look at how you can use ASP.NET and CSS together.

Caveats

While this chapter includes a lot of great information about HTML and CSS, and how you can use them in conjunction with ASP.NET and Visual Studio 2008, there are several caveats you should be aware of.

First, because there is no way that a single chapter can begin to cover the entire breadth of HTML and CSS, if you are looking for an in-depth discussion of these topics, you can check out the Wrox title *Beginning CSS: Cascading Style Sheets for Web Design, 2nd Edition*, by Richard York (Wiley Publishing, Inc., 2007).

Second, because CSS is simply a specification, it is up to each browser vendor to actually interpret and implement that specification. As is so often the case in Web development, each browser has its own quirks in how it implements (or sometimes does not implement) different CSS features. While the samples in this chapter were tested on Internet Explorer 7, you should make sure to thoroughly test your Web sites in multiple browsers on multiple platforms in order to ensure that your CSS is rendering appropriately in each browser you are targeting.

Finally, the DOCTYPE you use in your Web pages can influence how the browser applies the CSS styles includes in your Web page. You should understand how different DOCTYPES influence the browser's rendering process in your Web page.

HTML and CSS Overview

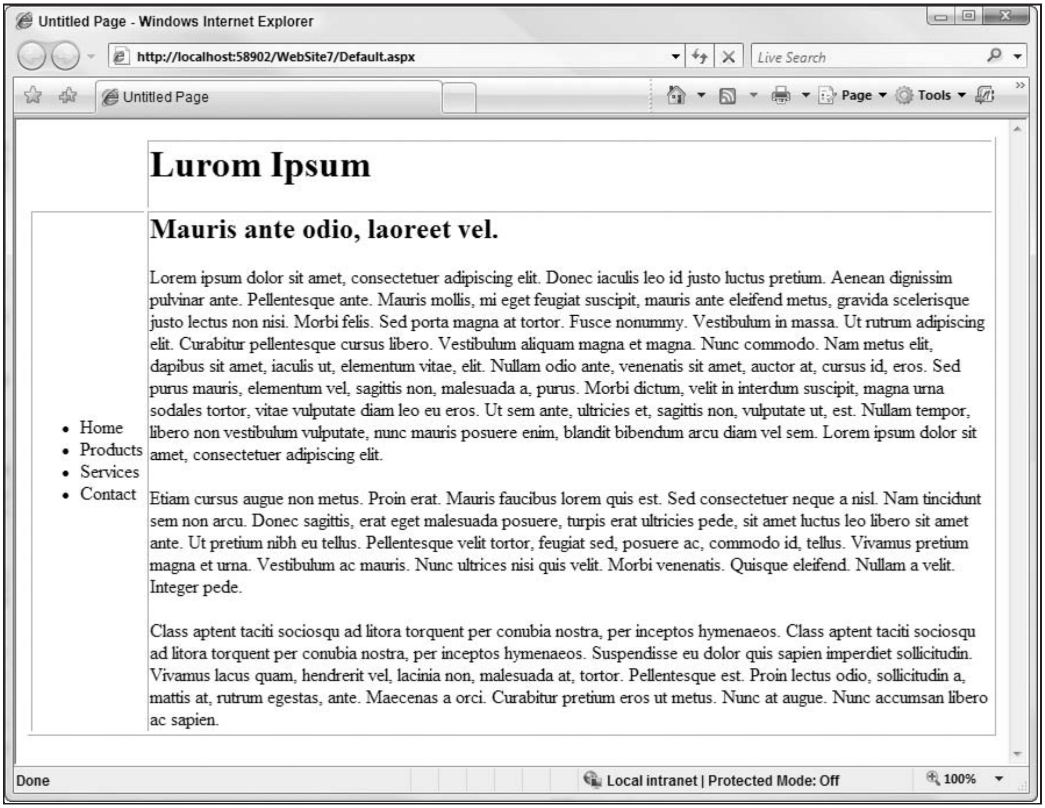
From the beginning of the Web, continuing to today, HTML serves as the primary mechanism for defining the content blocks of your Web page, and is the easiest way to define the layout of your Web page. HTML includes a variety of layout tags you can use, including Table, List, and Grouping elements. You can combine these elements to create highly complex layouts in your Web page. Figure 18-1 illustrates a single Web page that defines a basic layout using a variety of HTML elements.

While this layout is interesting, it lacks all but the most basic styling. To solve this problem, many developers would be tempted to start adding HTML-based formatting tags. For example, if I wanted to change the font and color of the text in the first paragraph, I might change its HTML to something like this:

```
<font face="Arial" Color="Maroon">
```

In fact, in the early days of Web design tools, this is what most of them generated when the user added styling to their Web pages, and for a while, using Font tags looks like a great solution to the problem of styling your Web pages.

Web developers and designers quickly learned, however, that using the Font tag quickly leads to a mess of spaghetti HTML, with font tags being splattered throughout the HTML. Imagine that if, in the previous example, you not only wanted to set the control and color, but some of the work needed to be bold, others needed to be a different color or font face, some a different font size, some underlined, and some displayed as superscript. Imagine how many font tags you would need then and how it would increase the weight of the Web page and decrease its maintainability. Using Font tags (and other style-related tags) meant that there was no longer a clear and clean separation between the structure and content of the Web page, but instead both were mashed together into a single complex document.


Untitled Page - Windows Internet Explorer
http://localhost:58902/WebSite7/Default.aspx
Live Search
Untitled Page
Page
Tools
Done
Local intranet | Protected Mode: Off
100%

Lurom Ipsum

Mauris ante odio, laoreet vel.

- Home
- Products
- Services
- Contact

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec iaculis leo id justo luctus pretium. Aenean dignissim pulvinar ante. Pellentesque ante. Mauris mollis, mi eget feugiat suscipit, mauris ante eleifend metus, gravida scelerisque justo lectus non nisi. Morbi felis. Sed porta magna at tortor. Fusce nonummy. Vestibulum in massa. Ut rutrum adipiscing elit. Curabitur pellentesque cursus libero. Vestibulum aliquam magna et magna. Nunc commodo. Nam metus elit, dapibus sit amet, iaculis ut, elementum vitae, elit. Nullam odio ante, venenatis sit amet, auctor at, cursus id, eros. Sed purus mauris, elementum vel, sagittis non, malesuada a, purus. Morbi dictum, velit in interdum suscipit, magna urna sodales tortor, vitae vulputate diam leo eu eros. Ut sem ante, ultricies et, sagittis non, vulputate ut, est. Nullam tempor, libero non vestibulum vulputate, nunc mauris posuere enim, blandit bibendum arcu diam vel sem. Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Etiam cursus augue non metus. Proin erat. Mauris faucibus lorem quis est. Sed consectetuer neque a nisl. Nam tincidunt sem non arcu. Donec sagittis, erat eget malesuada posuere, turpis erat ultricies pede, sit amet luctus leo libero sit amet ante. Ut pretium nibh eu tellus. Pellentesque velit tortor, feugiat sed, posuere ac, commodo id, tellus. Vivamus pretium magna et urna. Vestibulum ac mauris. Nunc ultrices nisi quis velit. Morbi venenatis. Quisque eleifend. Nullam a velit. Integer pede.

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Suspendisse eu dolor quis sapien imperdiet sollicitudin. Vivamus lacus quam, hendrerit vel, lacinia non, malesuada at, tortor. Pellentesque est. Proin lectus odio, sollicitudin a, mattis at, rutrum egestas, ante. Maecenas a orci. Curabitur pretium eros ut metus. Nunc at augue. Nunc accumsan libero ac sapien.

Figure 18-1

Introducing CSS

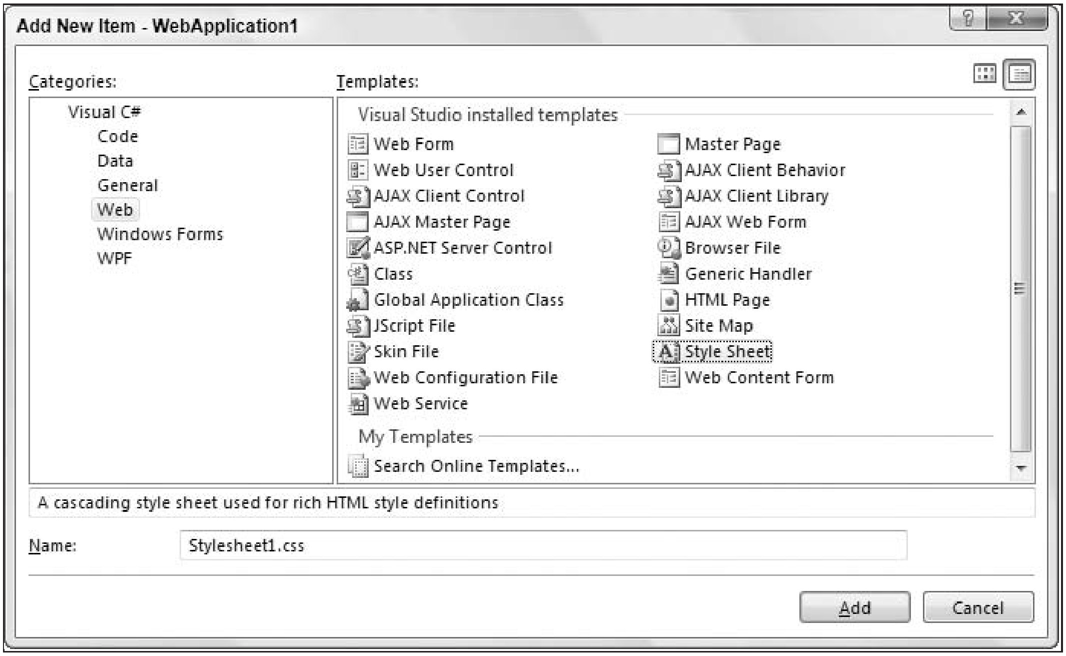
The introduction of CSS to the Web development and design world brought it back to a clean and elegant solution for styling Web pages. CSS meant a style could be defined in a single location for the entire Web site, and simply referenced on the elements requiring the style. Using CSS brought back the logic separation between Web page content and the styles used to display it.

Creating Style Sheets

Like HTML, CSS is an interpreted language. When a Web page request is processed by a Web server, the server's response can include style sheets, which are simply collections of CSS instructions. The style sheets can be included in the servers' response in three different ways: through external style sheet files, internal style sheets embedded directly in the Web page, or inline styles sheets.

External Style Sheets

External Style Sheets are collections of CSS styles stored outside of the Web pages that will use them — generally files using the .css extension. Visual Studio makes it simple to add external style sheet files to your application by including a Style Sheet file template in the Add New Item dialog box, as shown in Figure 18-2.

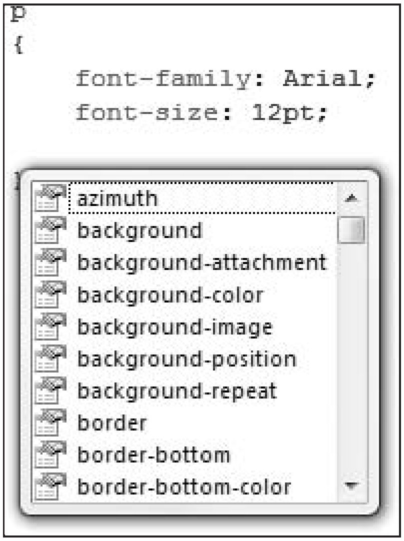

Add New Item - WebApplication1
Categories:
Visual C#
Code
Data
General
Web
Windows Forms
WPF
Templates:
Visual Studio installed templates
Web Form
Web User Control
AJAX Client Control
AJAX Master Page
ASP.NET Server Control
Class
Global Application Class
JScript File
Skin File
Web Configuration File
Web Service
Master Page
AJAX Client Behavior
AJAX Client Library
AJAX Web Form
Browser File
Generic Handler
HTML Page
Site Map
Style Sheet
Web Content Form
My Templates
Search Online Templates...
Add
Cancel

A cascading style sheet used for rich HTML style definitions

Name: Stylesheet1.css

Figure 18-2

Once the Style Sheet is created by Visual Studio, it's easy to insert new styles. Visual Studio even gives you CSS IntelliSense when working with styles in the document, as shown in Figure 18-3.



```
p
{
    font-family: Arial;
    font-size: 12pt;
```

azimuth
background
background-attachment
background-color
background-image
background-position
background-repeat
border
border-bottom
border-bottom-color

Figure 18-3

External style sheets are linked into Web pages using the HTML `<link>` tag. A single Web page can contain multiple style sheet references, as shown in Listing 18-1.

Listing 18-1: Using external style sheets in a Web page

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CSS Inheritance Sample</title>
    <link href="SampleStyles.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <form id="form1" runat="server">
       <div>Lorum Ipsum</div>
    </form>
</body>
</html>
```

You can add multiple `link` tags to a single Web page in order to link several different style sheets into the page. You can also use the CSS `import` statement directly in your style sheet to actually link multiple style sheets together.

```
@import url("layout.css");
```

Using the `import` statement has the advantage that you can alter the style sheets linked together without having to modify every Web page in your site. Instead, you can simply link each page to a master external style sheet, which in turn will use the `import` statement to link in other external style sheets. Note that older browsers may not understand this syntax and will simply ignore the command.

Using external style sheets in your Web site offers several advantages. First, because external style sheets are kept outside of the Web pages in your site, it is easier to add a link tag to all of your Web pages rather than trying to manage the styles directly in each page. This also makes maintenance easier because, should you decide to update the style of your Web site in the future, you have a single location in which styles are kept. Finally, using external style sheets can also help the performance of your Web site by allowing the browser to take advantage of its caching capabilities. Like other files downloaded by the browser, the style sheets will be cached on the client once they have been downloaded.

Internal Style Sheets

Internal style sheets are collections of CSS styles that are stored internally in a single Web page. The styles are located inside of the HTML `<style>` tag, which is generally located in the `<head>` section of the Web page. An example of internal style sheets is shown in Listing 18-2.

Listing 18-2: Using internal style sheets in a Web page

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CSS Inheritance Sample</title>
    <style type="text/css">
```

Continued

```
            div
            {
                 font-family:Arial;
            }
        </style>
    </head>
    <body>
        <form id="form1" runat="server">
           <div>Lorum Ipsum</div>
        </form>
    </body>
    </html>
```

It is important when you create internal style sheets that when you create style blocks, you make sure to include the `type` attribute with the style tag so the browser knows how to properly interpret the block. Additionally, as with external style sheets, Visual Studio also gives you IntelliSense support to make it easy for you to add properties.

Inline Styles

Inline styles are CSS styles that are applied directly to an individual HTML element using the element's `Style` attribute which is available on most HTML elements. An example of inline styles is shown in Listing 18-3.

Listing 18-3: Using inline styles in a Web page

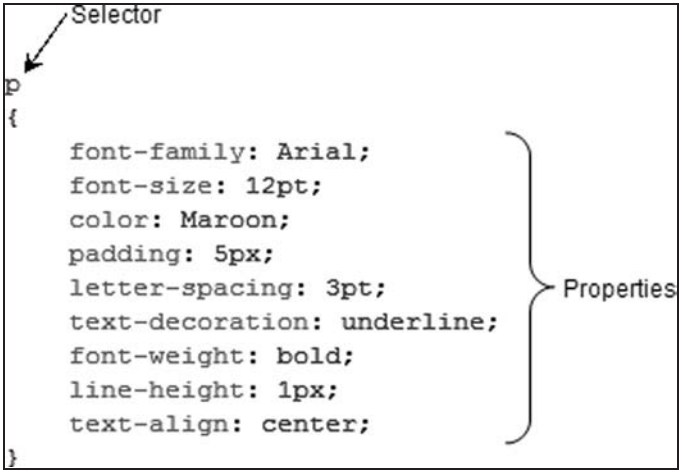
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CSS Inheritance Sample</title>
</head>
<body>
    <form id="form1" runat="server">
       <div style="font-family:Arial;">Lorum Ipsum</div>
    </form>
</body>
</html>
```

CSS Rules

Regardless of how they are stored, once CSS styles are sent from the server to the client, the browser is responsible for parsing the styles and applying them to the appropriate HTML elements in the Web page. If a style is stored in either an external or internal style sheet, the styles will be defined as a CSS rule. Rules are what the browser uses to determine what styling to apply, and to what HTML elements it should.

Inline styles do not need to be defined as a rule because they are automatically applied to the element they are included with. Therefore, the browser does not need to select the elements to apply it to.

A rule is made up of two parts, the Selector and the Properties. Figure 18-4 shows an example of a CSS rule.



```
p
{
    font-family: Arial;
    font-size: 12pt;
    color: Maroon;
    padding: 5px;
    letter-spacing: 3pt;
    text-decoration: underline;
    font-weight: bold;
    line-height: 1px;
    text-align: center;
}
```

Figure 18-4

Selectors

The Selector is the portion of the rule that dictates exactly how the Web browser should select the elements to apply the style to. CSS includes a variety of types of selectors, each of which defines a different element selection technique

Universal Selectors

The Universal Selector indicates that the style should apply to any element in the Web page. The sample that follows shows a Universal Selector, which would change the font of any element that supports the font-family property to Arial.

```
*
{
    font-family:Arial;
}
```

Type Selectors

The Type Selector allows you to create a style that applies to a specific type of HTML element. The style will then be applied to all elements of that type in the Web page. The following sample shows a Type Selector configured for the HTM paragraph tag, which will change the font family of all `<p>` tags in the Web page to Arial.

```
p
{
    font-family:Arial;
}
```

Descendant Selectors

Descendant Selectors allow you to create styles that target HTML elements that are descendants of a specific type of element. The following sample demonstrates a style that will be applied to any `<span>` tag that is a descendant of a `<div>`.

```
div span
{
    font-family:Arial;
}
```

Child Selectors

The Child Selector is similar to the Descendant Selector except unlike the Descendant Selector, which searches the entire descendant hierarchy of an element, the Child Selector restricts its element search to only those elements who are direct children of the parent element. The following code shows a modification of the Descendant Selector, making it a Child Selector.

```
div > span
{
    font-family:Arial;
}
```

Attribute Selectors

An Attribute Selector allows you to define a style that is applied to elements based on the existence of element attributes rather than the actual element name. For example, the following sample creates a style that is applied to any element in the Web page that has the `href` attribute set.

```
*[href]
{
    font-family:Arial;
}
```

Note that Attribute Selectors are not supported by Internet Explorer 6 or earlier, or by the Visual Studio 2008 design surface.

Adjacent Selectors

Adjacent selectors allow you to select HTML elements that are immediately adjacent to another element type. For example, in an unordered list, you may want to highlight the first list item and then have all the following items use a different style. You can use an Adjacent Selector to do this, which is shown in the following sample:

```
li
{
    font-size:xx-large;
}

li+li
{
    font-size:medium;
}
```

In this sample, a default Type Selector has been created for the list item element (`<li>`), which will change the font size of the text in the element to extra, extra large. However a second Adjacent Selector has been created, which will override the Type Selector for all list items after the first, changing the font size back to normal.

Class Selectors

Class Selectors are a special type of CSS selector that allows you to apply a style to any element with a specific Class name. The Class name is defined in HTML using the Class attribute, which is present on almost every element. Class Selectors are distinguished from other Selector types by prefixing them with a single period (.).

```
.title
{
    font-size:larger;
    font-weight:bold;
}
```

This CSS rule would then be applied to any element whose class attribute value matched the rule name, an example of which is shown here:

```
<div class="title">Lorum Ipsum</div>
```

When creating Class Selectors, note that the class name may not begin with a numeric character. Also, CSS class names can contain only alphanumeric characters. Spaces, symbols and even underscores are not allowed. Finally, you should make sure that you match the casing of your class name when using it in the HTML. While CSS itself is not case sensitive, some HTML DocTypes dictate that the `class` and `id` attributes be treated as case sensitive.

ID Selectors

ID Selectors are another special type of CSS Selector that allows you to create styles that target elements with specific ID values. ID Selectors are distinguished from other Selector types by prefixing them with a hash mark (#).

```
#title
{
    font-size:larger;
    font-weight:bold;
}
```

This CSS rule would be applied to any element whos `id` attribute value matched the Rule name, an example of which is shown here:

```
<div id="title">Lorum Ipsum</div>
```

Pseudo Classes

CSS also includes a series of pseudo class selectors that give you additional options in creating Style rules. Pseudo classes can be added to other selectors to allow you to create more complex rules.

First Child Pseudo Class

The first-child pseudo class allows you to indicate that the rule should select the first child element M of an element N. The following is an example of using the first-child pseudo class:

```
#title p:first-child
{
    font-size:xx-small;
}
```

The Rule defined above states that the style should be applied to the first paragraph tag found within any element with an `id` attribute value of `title`. In the following HTML, that means that the text `First Child` would have the style applied to it:

```
<div id="title">Lorum <p>First Child</p><p>Second Child</p> Ipsum</div>
```

Note that the Visual Studio 2008 design surface does not support the first-child pseudo class; therefore, even though the style may be rendered properly in the browser, you may not get an accurate preview on the design surface.

Link Pseudo Classes

CSS includes a number of pseudo classes specifically related to anchor tags. These special pseudo classes allow you to define styles for the different states of an anchor tag.

```
a:link
{
    color:Maroon;
}
a:visited
{
    color:Silver;
}
```

In this sample, two rules have been created, the first of which applies a style to the unvisited links in a page, while the second applies a different style to the visited links.

Dynamic Pseudo Classes

The dynamic pseudo classes are special CSS classes that are applied by the browser based on actions performed by the end user such as hovering over an element, activating an element, or giving an element focus.

```
a:hover
{
    color:Maroon;
}
a:active
{
    color:Silver;
}
a:focus
```

```
{
    color:Olive;
}
```

While the sample demonstrates the use of the dynamic pseudo classes with the anchor tag, they can be used with any HTML element. Note, however, that support for the dynamic pseudo classes in different browsers varies.

Language Pseudo Class

The language pseudo class allows you to define specific Rules based on the end user's language settings.

```
:lang(de)
{
    quotes: '≪' '≫' '\2039' '\203A'
}
```

In this sample, the `lang` pseudo class is used to set the quotes for a Web page that is German. The `lang` pseudo class is not supported by IE 7.

Pseudo Elements

CSS also includes several pseudo elements, which allow you to make selections of items in the Web page that are not true elements.

The pseudo elements available are: first-line, first letter, before, and after. The following samples demonstrate the use of these.

```
p:first-line
{
    font-style:italic;
}
p:first-letter
{
    font-size:xx-large;
}
```

The pseudo first-line and first-letter elements allow you to apply special styling to the first line and first letter of a content block.

```
p:before
{
    content: url(images/quote.gif);
}
p:after
{
    content: '<<end>>';
}
```

The pseudo `before` and `after` elements allow you to insert content before or after the targeted element, in this case a paragraph element. The content you insert can be a URL, string, Quote character, counter, or the value of an attribute of the element.

Selector Grouping

When creating CSS rules, CSS allows you group several selectors together into a single rule. The following sample demonstrates a single rule that combines three type selectors:

```
h1, h2, h3
{
    color:Maroon;
}
```

This rule then results in the fore-color of the text content of any h1, h2 or h3 tag being maroon.

Selector Combinations

CSS also allows you to combine multiple Selector types. For example, you can create Class selectors that target specific HTML elements in addition to matching the `Class` attribute value.

Listing 18-4: Combining multiple Selector types in a single CSS rule

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">

    <style type="text/css">
        .title
        {
            font-family:Courier New;
        }

        div.title
        {
            font-family:Arial;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
        <p class="title">Lorum Ipsum</p>
        <div class="title">Lorum Ipsum</div>
    </form>
</body>
</html>
```

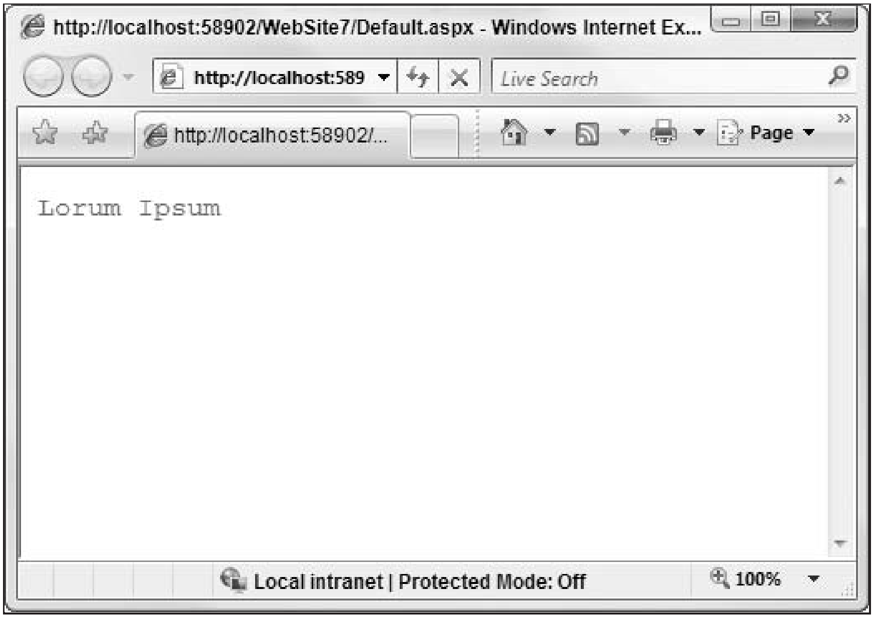
Merged Styles

CSS also merges styles when several style rules are defined that apply to a given HTML element. For example, in the sample code that follows, a Class Selector and a Type Selector are defined. Both of these selectors apply to the paragraph element in the HTML. When the browser interprets the styles, it will merge both onto the element.

Listing 18-5: Merging styles from multiple rules onto a single element

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">

    <style type="text/css">
         .title
        {
            font-family:Courier New;
        }

        p
        {
            color:Green;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
        <p class="title">Lorum Ipsum</p>
    </form>
</body>
</html>
```

As you can see in Figure 18-5, both the font and the color of the single paragraph element have been styled, even though there were two separate style rules that defined the style.


http://localhost:58902/WebSite7/Default.aspx - Windows Internet Ex...
http://localhost:589
Live Search
http://localhost:58902/...
Page
Local intranet | Protected Mode: Off
100%

Lorum Ipsum

Figure 18-5

You can also merge multiple styles by defining multiple rules using different Selector types. If a single HTML element matches all of the rules, the styles from each rule will be merged together. Listing 18-6 shows an example where a single element matches multiple rules.

Listing 18-6: Multiple Selector matches on a single element

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CSS Inheritance Sample</title>
    <style type="text/css">
        p
        {
            font-family:Arial;
            color:Blue;
        }

        p#book
        {
            font-size:xx-large;
        }

        p.title
        {
            font-family: Courier New;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
       <p id="book" class="title" style="letter-spacing:5pt;">Lorum Ipsum</p>
    </form>
</body>
</html>
```

In this case, because the paragraph tag defines the `id`, `class`, and `style` attributes, each of the Style rules match; therefore, each of their styles get merged onto the element.

Finally the class attribute itself can be used to merge multiple styles onto the same element. The class attribute allows you to specify multiple class names in a space-delimited string.

Listing 18-7: Assigning multiple Class Selectors to a single element

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CSS Inheritance Sample</title>
    <style type="text/css">
        p.title
        {
            font-family: Courier New;
            letter-spacing:5pt;
        }

        p.summer
        {
            color:Blue;
        }

        p.newproduct
        {
            font-weight:bold;
            color:Red;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
       <p class="title newproduct summer">Lorum Ipsum</p>
    </form>
</body>
</html>
```

In this case, the three classes — `title`, `summer`, and `newproduct` — have all been defined in the class attribute. This means that these three styles will be merged onto the paragraph element.

Note that, in this case, the order in which the CSS classes are defined in the internal stylesheet also influences how the styles are merged onto the paragraph tag. Even though the summer class is last in the list of classes defined in the Class attribute, the `newproduct` rule overrides the summer Rules color property because the `newproduct` Rule is defined after the summer rule in the internal style sheet.

CSS Inheritance

CSS includes the concept of style inheritance. This works because the browser views the different locations that a style can be defined in (external, internal, or inline) as a hierarchical structure. Figure 18-6 shows this inheritance by demonstrating how the `font-family` property of a paragraph type selector rule, defined in three different locations, could be overridden by other style rules.

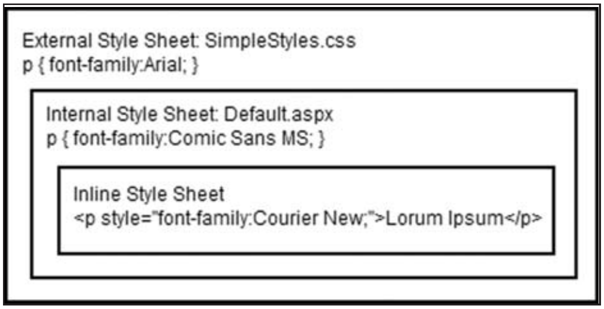

External Style Sheet: SimpleStyles.css
p { font-family:Arial; }
Internal Style Sheet: Default.aspx
p { font-family:Comic Sans MS; }
Inline Style Sheet
<p style="font-family:Courier New;">Lorum Ipsum</p>

Figure 18-6

As you can see from the figure, the rule of thumb is that the closer the style definition is to the element it applies to, the more precedence it will take. In this case, the paragraph text would ultimately be displayed using the Courier New font family because that is defined in the inline style.

Inheritance not only applies to styles kept in separate file locations, but also applies to styles within the same location, which means that sometimes you also need to think about the order in which you define your styles. For example, Listing 18-8 shows a style sheet that contains two Type Selectors, both targeting the paragraph element, both setting the `font-family` style property. Obviously both of these cannot be applied to the same element, so CSS simply chooses the Selector that is closest to the paragraph tags.

Listing 18-8: Using style overriding within the same Internal style sheet

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">

    <style type="text/css">
        p
        {
            font-family:Arial;
        }

        p
        {
            font-family: Courier New;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
       <p>Lorum Ipsum</p>
    </form>
</body>
</html>
```

Running this sample, you will see that the font applied is the Courier New font.

Note that you should be careful when combining styles from external style sheets and internal style sheets. Remember that the browser will ultimately choose the style that is defined closet to the specific elements. This means that as the browser begins to parse the Web page, internal styles defined before external styles are considered further away from the HTML elements. Thus, the browser will use the styles located in the external style sheet. If you plan on storing style rules in both internal and external style sheets, you should remember to include the external style sheets <link> tags before the internal style sheets <style> block in your Web page.

Element Layout and Positioning

CSS is useful not only for styling elements in a page, but also for positioning elements as well. CSS actually gives you a much more flexible system for positioning elements than HTML itself. CSS bases the positioning of elements in a Web page on something called the *box model*. Once an element's box behavior has been determined, it can be positioned using several different techniques.

The CSS Box Model

A core element of positioning in CSS is the box model. The box model defines how every element in HTML is treated by the browser as a rectangular box. The box comprises different parts, including margins, padding, borders, and content. Figure 18-7 shows how all of these elements are combined to form the box.

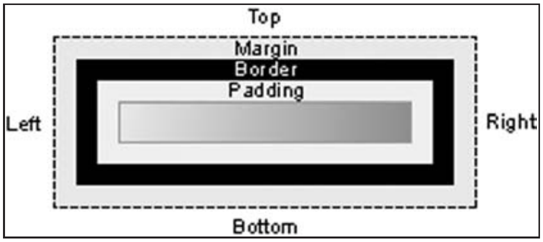


Figure 18-7

All of the separate elements that make up the box can influence its position within the Web page, and unless otherwise specified, each is given a default value of zero. The height and width of the element is equal to the height and width of the outer edge of the margin, which as you can see in the previous image, is not necessarily the height and width of the content.

HTML provides you with two different types of boxes, the block box and the inline box. Block boxes are typically represented by tags such as `<p>`, `<div>`, or `<table>`. For block boxes, the containing block is used to determine the position of its child blocks. Additionally, block boxes can contain only inline or block boxes, but not both.

Listing 18-9 shows an example of a page containing a single parent block and two child block elements.

Listing 18-9: Creating block box elements

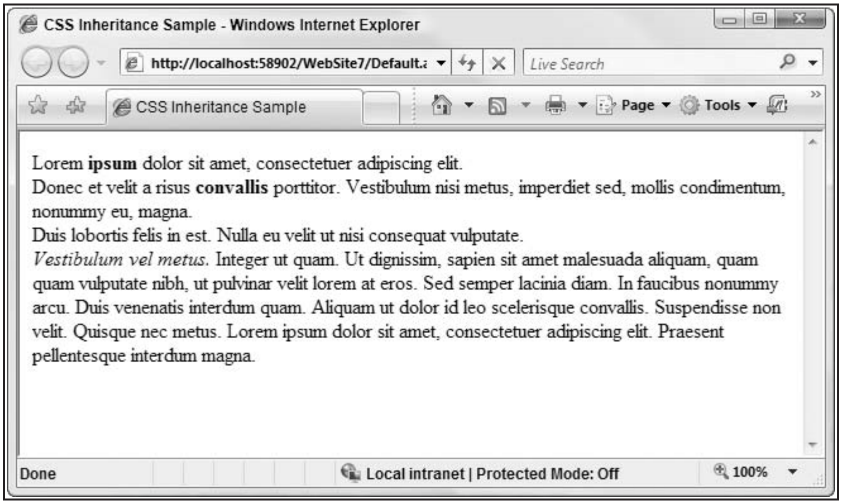
```
<div>
    Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
    <div>
        Donec et velit a risus convallis porttitor.
       Vestibulum nisi metus, imperdiet sed, mollis condimentum, nonummy eu, magna.\
    </div>
   <div>Duis lobortis felis in est. Nulla eu velit ut nisi consequat vulputate.</div>
    Vestibulum vel metus. Integer ut quam. Ut dignissim, sapien sit amet
malesuada aliquam,
    quam quam vulputate nibh, ut pulvinar velit lorem at eros.
Sed semper lacinia diam. In
    faucibus nonummy arcu. Duis venenatis interdum quam. Aliquam ut dolor id leo
    scelerisque convallis. Suspendisse non velit. Quisque nec metus.
Lorem ipsum dolor sit
    amet, consectetuer adipiscing elit. Praesent pellentesque interdum magna.
</div>
```

The second box type is the inline box. Inline boxes are typically represented by tags such as `B`, `I`, and `SPAN` well as and actual text and content. Listing 18-10 shows how the previous listing can be modified to include inline boxes.

Listing 18-10: Creating inline box elements

```
<div>
    Lorem <b>ipsum</b> dolor sit amet, consectetuer adipiscing elit.
    <div>
        Donec et velit a risus <b>convallis</b> porttitor.
        Vestibulum nisi metus, imperdiet sed, mollis condimentum, nonummy eu, magna.
    </div>
    <div>Duis lobortis felis in est.
        <span>Nulla eu velit ut nisi consequat vulputate.</span>
    </div>
    <i>Vestibulum vel metus.</i> Integer ut quam. Ut dignissim, sapien sit
amet malesuada
    aliquam, quam quam vulputate nibh, ut pulvinar velit lorem at eros.
Sed semper lacinia
    diam. In faucibus nonummy arcu. Duis venenatis interdum quam.
Aliquam ut dolor id leo
    scelerisque convallis. Suspendisse non velit. Quisque nec metus.
Lorem ipsum dolor sit
    amet, consectetuer adipiscing elit. Praesent pellentesque interdum magna.
</div>
```

Rendering this page results in each block beginning a new line. Figure 18-8 shows the markup rendered in the browser.

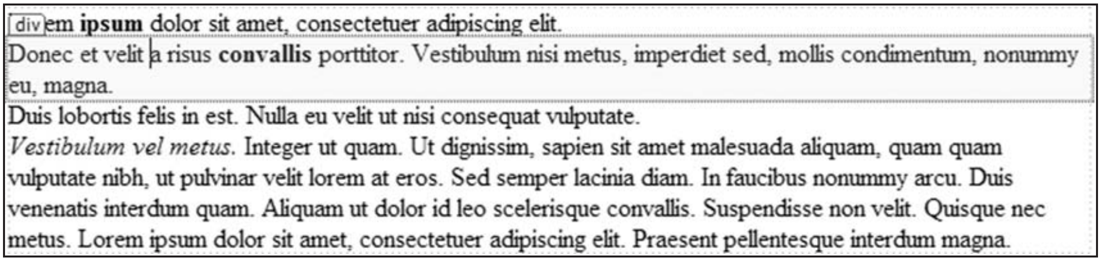

CSS Inheritance Sample - Windows Internet Explorer
http://localhost:58902/WebSite7/Default.a
Live Search
CSS Inheritance Sample
Page
Tools
Done
Local intranet | Protected Mode: Off
100%

Lorem **ipsum** dolor sit amet, consectetuer adipiscing elit.
Donec et velit a risus **convallis** porttitor. Vestibulum nisi metus, imperdiet sed, mollis condimentum, nonummy eu, magna.
Duis lobortis felis in est. Nulla eu velit ut nisi consequat vulputate.
Vestibulum vel metus. Integer ut quam. Ut dignissim, sapien sit amet malesuada aliquam, quam quam vulputate nibh, ut pulvinar velit lorem at eros. Sed semper lacinia diam. In faucibus nonummy arcu. Duis venenatis interdum quam. Aliquam ut dolor id leo scelerisque convallis. Suspendisse non velit. Quisque nec metus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent pellentesque interdum magna.

Figure 18-8

The Visual Studio design surface can help you get a clear picture of the layout of a `div` as well. When you select an individual div element, the design surface highlights the selected element, as shown in Figure 18-9.

At the beginning of this section, I stated that a block will always container either inline or block boxes, but it’s interesting to note that in this case, because the first line of text contains an inline box, and the next contains a block box, it looks like the parent div is violating that rule. However, what is actually happening is that the browser automatically adds an anonymous block box around the first line of text when the page is rendered. Figure 18-10 highlights the block boxes as the browser sees them.



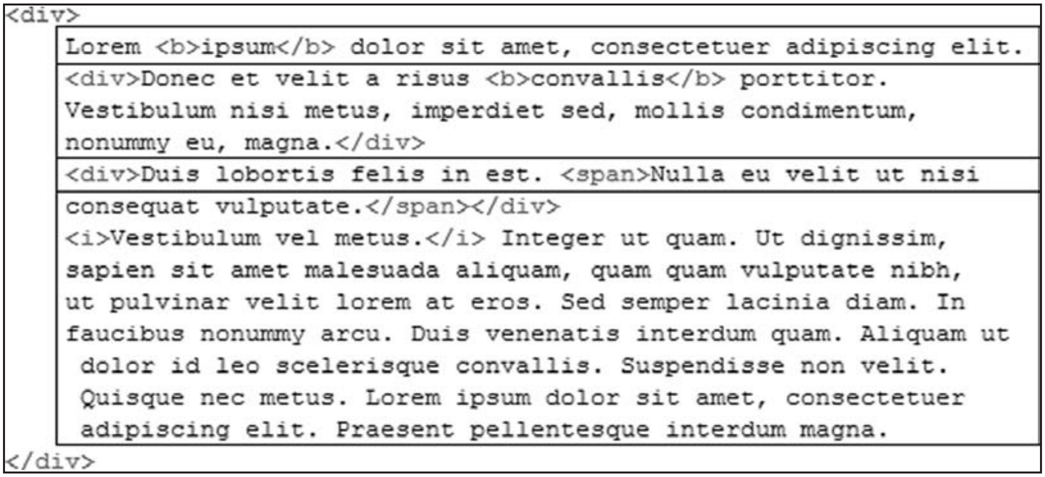
em **ipsum** dolor sit amet, consectetuer adipiscing elit.

Donec et velit a risus **convallis** porttitor. Vestibulum nisi metus, imperdiet sed, mollis condimentum, nonummy eu, magna.

Duis lobortis felis in est. Nulla eu velit ut nisi consequat vulputate.

Vestibulum vel metus. Integer ut quam. Ut dignissim, sapien sit amet malesuada aliquam, quam quam vulputate nibh, ut pulvinar velit lorem at eros. Sed semper lacinia diam. In faucibus nonummy arcu. Duis venenatis interdum quam. Aliquam ut dolor id leo scelerisque convallis. Suspendisse non velit. Quisque nec metus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent pellentesque interdum magna.

Figure 18-9



```
<div>
    Lorem <b>ipsum</b> dolor sit amet, consectetuer adipiscing elit.
    <div>Donec et velit a risus <b>convallis</b> porttitor.
    Vestibulum nisi metus, imperdiet sed, mollis condimentum,
    nonummy eu, magna.</div>
    <div>Duis lobortis felis in est. <span>Nulla eu velit ut nisi
    consequat vulputate.</span></div>
    <i>Vestibulum vel metus.</i> Integer ut quam. Ut dignissim,
    sapien sit amet malesuada aliquam, quam quam vulputate nibh,
    ut pulvinar velit lorem at eros. Sed semper lacinia diam. In
    faucibus nonummy arcu. Duis venenatis interdum quam. Aliquam ut
     dolor id leo scelerisque convallis. Suspendisse non velit.
     Quisque nec metus. Lorem ipsum dolor sit amet, consectetuer
     adipiscing elit. Praesent pellentesque interdum magna.
</div>
```

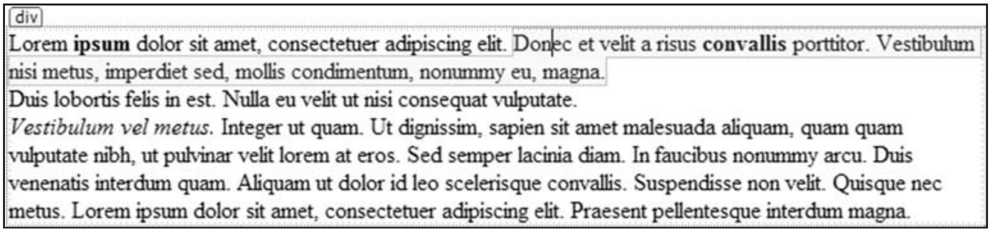
Figure 18-10

You can explicitly set which box behavior an element will exhibit by using the `position` attribute. For example, setting the `position` property on the second div, as shown here, results in the layout of the content changing.

```
<div style="display:inline;">Donec et velit a risus <b>convallis</b> porttitor.
Vestibulum nisi metus, imperdiet sed, mollis condimentum, nonummy eu, magna.</div>
```

Figure 18-11 shows how adding this property changes the rendering of the markup on the Visual Studio design surface. You can see that now, rather than the element being displayed on a new line, its content is simply continued from the previous block.

You can also set the display property to `none` to completely remove the element from the Web page layout. If you have elements whose `display` property is set to `none`, or an element whose `visibility` property is set to `hidden`, Visual Studio gives you the option of showing or hiding these elements on its design surface.



Lorem **ipsum** dolor sit amet, consectetuer adipiscing elit. Donec et velit a risus **convallis** porttitor. Vestibulum nisi metus, imperdiet sed, mollis condimentum, nonummy eu, magna.
Duis lobortis felis in est. Nulla eu velit ut nisi consequat vulputate.
Vestibulum vel metus. Integer ut quam. Ut dignissim, sapien sit amet malesuada aliquam, quam quam vulputate nibh, ut pulvinar velit lorem at eros. Sed semper lacinia diam. In faucibus nonummy arcu. Duis venenatis interdum quam. Aliquam ut dolor id leo scelerisque convallis. Suspendisse non velit. Quisque nec metus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent pellentesque interdum magna.

Figure 18-11

As shown in Figure 18-12, there are two options on the View menu that allow you to toggle the design surface visibility of elements with these properties set.

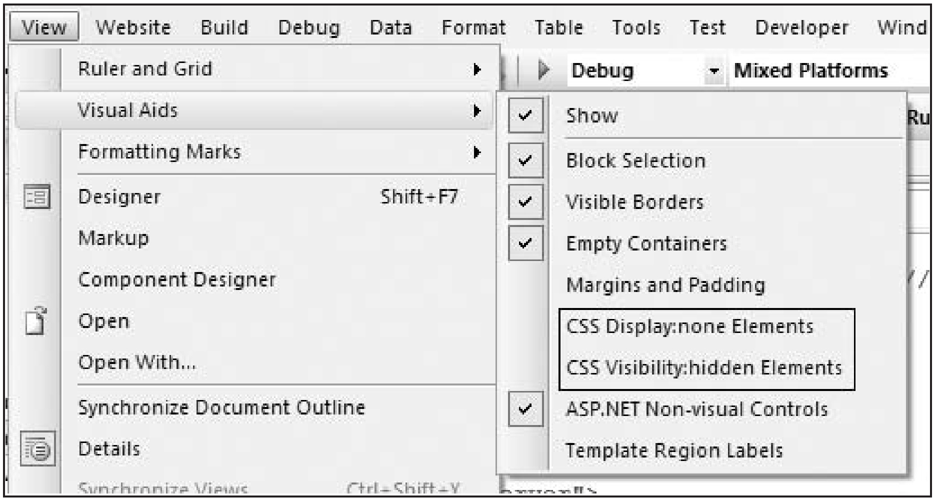

View
Website
Build
Debug
Data
Format
Table
Tools
Test
Developer
Wind
Ruler and Grid
Visual Aids
Formatting Marks
Designer Shift+F7
Markup
Component Designer
Open
Open With...
Synchronize Document Outline
Details
Synchronize Views Ctrl+Shift+Y
Debug
Mixed Platforms
✓ Show
✓ Block Selection
✓ Visible Borders
✓ Empty Containers
Margins and Padding
CSS Display:none Elements
CSS Visibility:hidden Elements
✓ ASP.NET Non-visual Controls
Template Region Labels

Figure 18-12

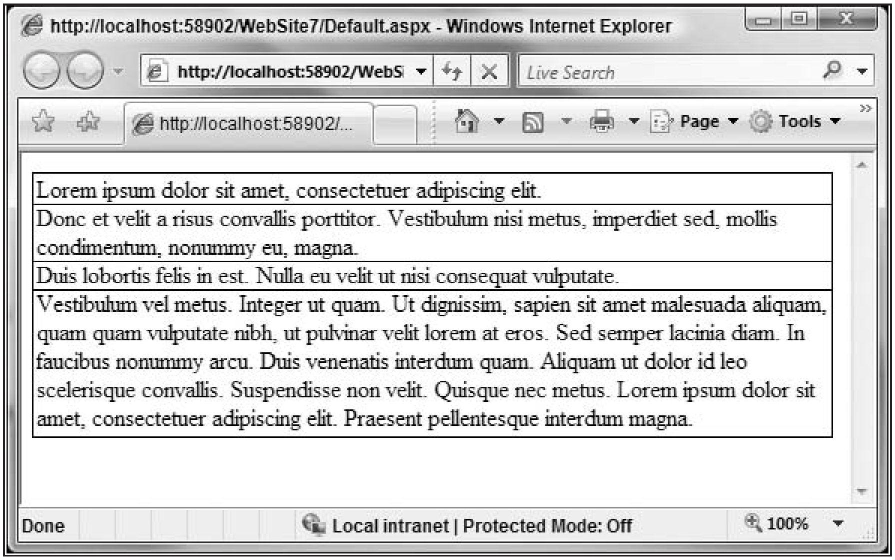
Positioning CSS Elements

CSS provides you with three primary positioning mechanisms: Normal, Absolute, and Relative. Each type offers a different behavior you can use to lay out the elements in your page. To specify the type of layout behavior you want an element use, you can set the CSS `position` property. Each element can have its own position property set, allowing you to use multiple positioning schemes within the same Web page.

Normal Positioning

Using Normal positioning, block items flow vertically, and inline items flow horizontally, left to right. This is the default behavior, and is used when no other value is provided for the position property. Figure 18-13 demonstrates the layout of four separate blocks using Normal positioning.

As you can see, each block item flows vertically as expected.


http://localhost:58902/WebSite7/Default.aspx - Windows Internet Explorer
http://localhost:58902/WebS
Live Search
http://localhost:58902/...
Page
Tools
Done
Local intranet | Protected Mode: Off
100%

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Donc et velit a risus convallis porttitor. Vestibulum nisi metus, imperdiet sed, mollis condimentum, nonummy eu, magna.

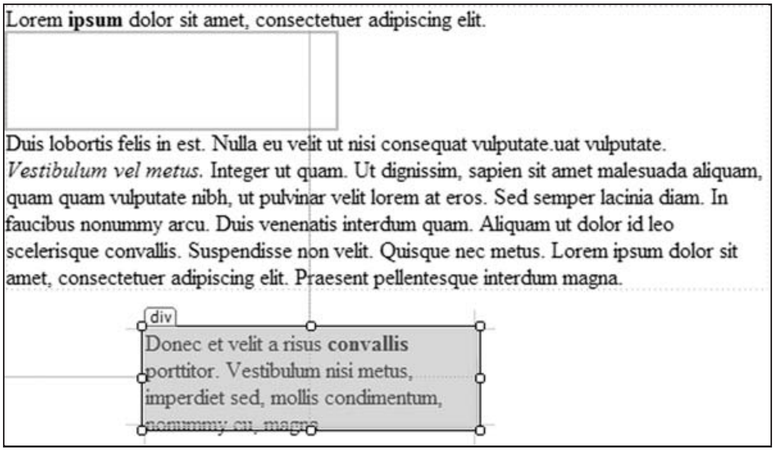
Duis lobortis felis in est. Nulla eu velit ut nisi consequat vulputate.

Vestibulum vel metus. Integer ut quam. Ut dignissim, sapien sit amet malesuada aliquam, quam quam vulputate nibh, ut pulvinar velit lorem at eros. Sed semper lacinia diam. In faucibus nonummy arcu. Duis venenatis interdum quam. Aliquam ut dolor id leo scelerisque convallis. Suspendisse non velit. Quisque nec metus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent pellentesque interdum magna.

Figure 18-13

Relative Positioning

Using Relative positioning, elements are initially positioned using Normal layout. The surrounding boxes are positioned and then the box is moved based on its offset properties: `top`, `bottom`, `left`, and `right`. Figure 18-14 shows the same content as in the prior section, but now the third block box has been styled to use Relative positioning. Visual Studio is helping you out by providing positioning lines for the block, showing you that its top offset is being calculated based on the normal top position of the block, and the left offset from the normal left position. Visual Studio even lets you visually position the block by grabbing the element's tag label and dragging it over the design surface.



Lorem **ipsum** dolor sit amet, consectetuer adipiscing elit.

Duis lobortis felis in est. Nulla eu velit ut nisi consequat vulputate.uat vulputate. *Vestibulum vel metus.* Integer ut quam. Ut dignissim, sapien sit amet malesuada aliquam, quam quam vulputate nibh, ut pulvinar velit lorem at eros. Sed semper lacinia diam. In faucibus nonummy arcu. Duis venenatis interdum quam. Aliquam ut dolor id leo scelerisque convallis. Suspendisse non velit. Quisque nec metus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent pellentesque interdum magna.

Donec et velit a risus **convallis** porttitor. Vestibulum nisi metus, imperdiet sed, mollis condimentum, nonummy eu, magna

Figure 18-14

As you position the element on the design surface, the element's top and left values are being updated. You will end up with an element looking something like this:

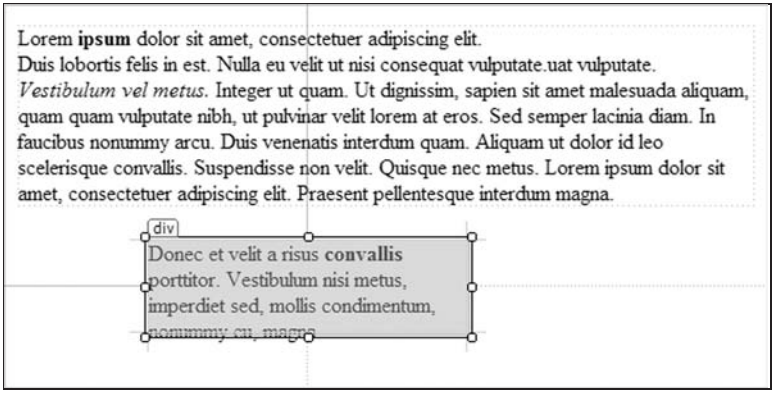
```
<div style="position: relative;top: 214px;left: 62px;width: 239px;height: 81px">
    Donec et velit a risus <b>convallis</b> porttitor. Vestibulum nisi metus,
    imperdiet sed, mollis condimentum, nonummy eu, magna.
</div>
```

If you are using relative positioning and have both left and right offsets defined, the right will generally be ignored.

Absolute Positioning

Absolute positioning works much like relative positioning, except instead of an element calculating its offset position based on its position in the normal positioning scheme, the offsets are calculated based on the position of its closest absolutely positioned ancestor. If no element exists, then the ancestor is the browser window itself.

Figure 18-15 shows how blocks using absolute positioning are displayed on the Visual Studio design surface. As you can see, unlike the display of the relative positioned element shown in the previous section, this time the positioning lines extend all the way to the edge of the design surface. This is because the block is using the browser window to calculate its offset.



Lorem **ipsum** dolor sit amet, consectetuer adipiscing elit.
Duis lobortis felis in est. Nulla eu velit ut nisi consequat vulputate.uat vulputate.
Vestibulum vel metus. Integer ut quam. Ut dignissim, sapien sit amet malesuada aliquam, quam quam vulputate nibh, ut pulvinar velit lorem at eros. Sed semper lacinia diam. In faucibus nonummy arcu. Duis venenatis interdum quam. Aliquam ut dolor id leo scelerisque convallis. Suspendisse non velit. Quisque nec metus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent pellentesque interdum magna.

Donec et velit a risus **convallis** porttitor. Vestibulum nisi metus, imperdiet sed, mollis condimentum, nonummy eu, magna

Figure 18-15

As with relative blocks, you can use the element's tag label to position the element on the page, and Visual Studio will automatically update the offset values. The block in Figure 18-15 would output an element that looks something like this:

```
<div style="position: absolute;top: 180px;left:94px;width:239px;height:81px;">
    Donec et velit a risus <b>convallis</b> porttitor. Vestibulum nisi metus,
    imperdiet sed, mollis condimentum, nonummy eu, magna.
</div>
```

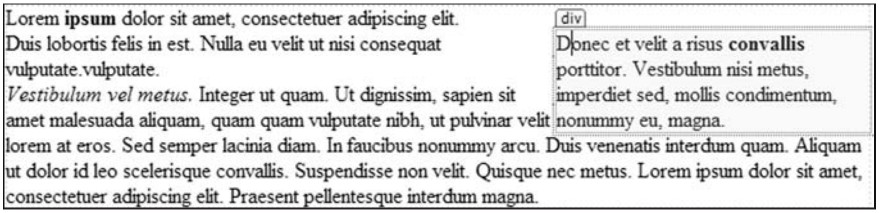
Floating Elements

Another option for controlling the position of elements using CSS is the `float` property. The float property allows you to float an element to the left or right side of a block. The floating block is positioned vertically as it would normally be in normal position, but horizontally shifted as far left or right as possible. Listing 18-11 demonstrates floating the same block used in previous samples in this section.

Listing 18-11: Floating a block element to the right

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
</head>
<body>
    <form id="form1" runat="server">
    <div id="asdas" class="werwer">
        Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
    </div>
    <div style="float:right;width: 236px;">
        Donec et velit a risus <b>convallis</b> porttitor. Vestibulum nisi metus,
        imperdiet sed, mollis condimentum, nonummy eu, magna.
    </div>
   <div>Duis lobortis felis in est. Nulla eu velit ut nisi consequat vulputate.</div>
    <div>
       Vestibulum vel metus. Integer ut quam. Ut dignissim, sapien sit amet malesuada
       aliquam, quam quam vulputate nibh, ut pulvinar velit lorem at eros. Sed semper
         lacinia diam. In faucibus nonummy arcu. Duis venenatis interdum quam.
Aliquam ut
        dolor id leo scelerisque convallis. Suspendisse non velit.
Quisque nec metus. Lorem
        ipsum dolor sit amet, consectetuer adipiscing elit.
Praesent pellentesque interdum
        magna.
    </div>

    </form>
</body>
</html>
```

The block has been modified to include the `float` property in its style. When this is done, Visual Studio correctly positions the element to the far right side of the page. This is shown in Figure 18-16.



Lorem **ipsum** dolor sit amet, consectetuer adipiscing elit.
Duis lobortis felis in est. Nulla eu velit ut nisi consequat vulputate.vulputate.
Vestibulum vel metus. Integer ut quam. Ut dignissim, sapien sit amet malesuada aliquam, quam quam vulputate nibh, ut pulvinar velit lorem at eros. Sed semper lacinia diam. In faucibus nonummy arcu. Duis venenatis interdum quam. Aliquam ut dolor id leo scelerisque convallis. Suspendisse non velit. Quisque nec metus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent pellentesque interdum magna.

Donec et velit a risus **convallis** porttitor. Vestibulum nisi metus, imperdiet sed, mollis condimentum, nonummy eu, magna.

Figure 18-16

The !important Attribute

As you saw earlier in this chapter, the browser will choose to apply the closest style to the element, which can mean that properties of other applied styles may be overridden. As with many other rules in CSS, this too is not absolute. CSS provides a mechanism to circumvent this called the `!important` attribute. Properties that have this attribute applied can prevent other CSS rules from overriding its value. Listing 18-8 showed how the `font-family` property can be overridden. You can see how the `!important` attribute works by modifying this sample to use the attribute. This is shown in Listing 18-12.

Listing 18-12: Using the !important attribute to control style overriding

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">

    <style type="text/css">
        p
        {
            font-family:Arial !important;
        }

        p
        {
            font-family: Courier New;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
       <p>Lorum Ipsum</p>
    </form>
</body>
</html>
```

In this case, rather than the paragraph being shown in Courier New, it will use the Arial font because it has been marked with the `!important` attribute.

Working with HTML and CSS in Visual Studio

Working with HTML and CSS to create compelling Web site designs can be quite daunting. Thankfully, Visual Studio provides you with a variety of tools that help simplify page layout and CSS management.

As you are probably already familiar with, Visual Studio includes a great WYSIWYG design surface. In prior versions of Visual Studio, this design surface used Internet Explorer to generate the design view, but with the release of Visual Studio 2008, Microsoft has completely rewritten the design surface to be completely independent of Internet Explorer.

In Visual Studio, when the Design View has focus, two additional menus become available: the Format menu and the Table menu. This is shown in Figure 18-17.

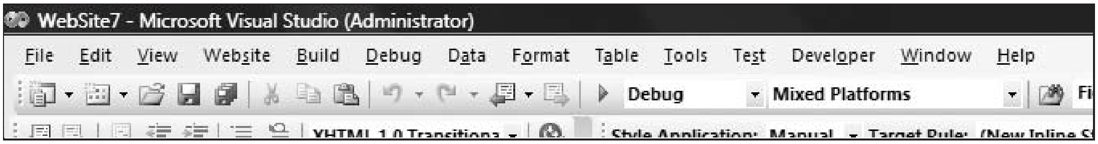
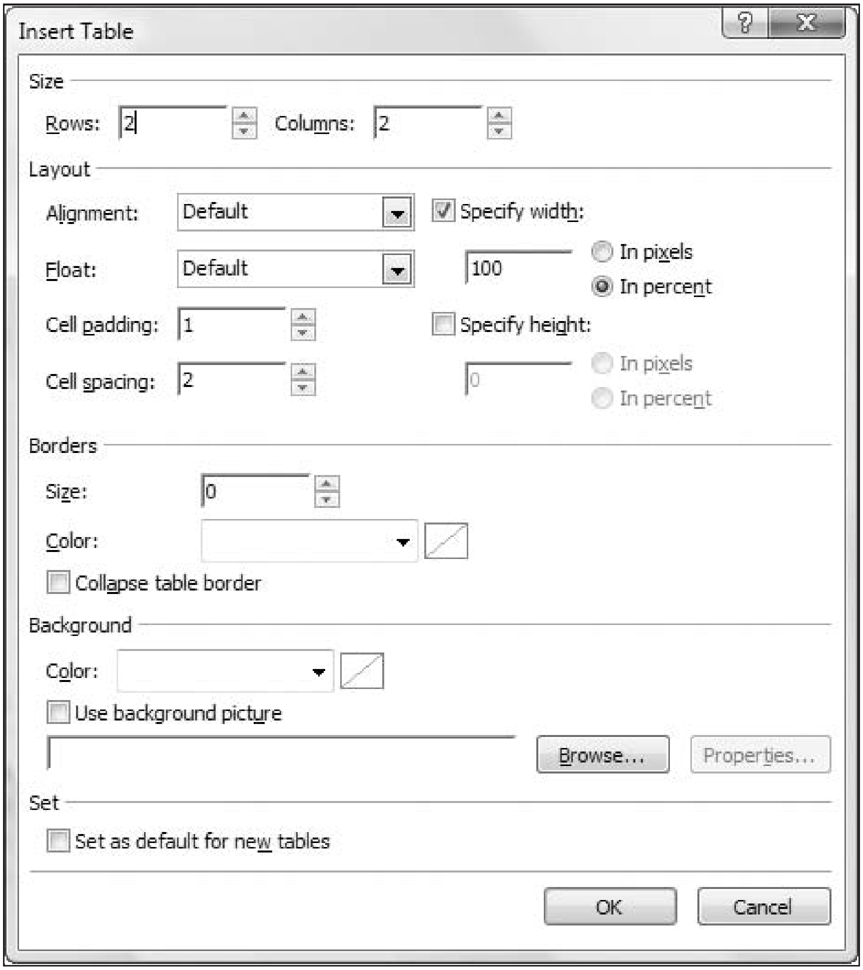

WebSite7 - Microsoft Visual Studio (Administrator)
File
Edit
View
Website
Build
Debug
Data
Format
Table
Tools
Test
Developer
Window
Help
Debug
Mixed Platforms
Fi
XHTML 1.0 Transitiona
Style Application: Manual
Target Rule: (New Inline St

Figure 18-17

The Table menu, as you might guess, includes a set of tools that allow you to Insert, Delete, Select, and Modify the HTML Tables in your Web page. Selecting the Insert Table option from the Table menu opens the Insert Table dialog box shown in Figure 18-18, which allows you to easily specify properties of your table. You can define the number of table rows and columns, the cell padding and spacing, and border attributes, and when you click OK, Visual Studio automatically generates the appropriate table HTML in your Web page.



Rows: 2
Columns: 2
Alignment: Default
Float: Default
Cell padding: 1
Cell spacing: 2
☑ Specify width: 100
○ In pixels
● In percent
☐ Specify height: 0
○ In pixels
○ In percent
Size: 0
Color:
☐ Collapse table border
Color:
☐ Use background picture
☐ Set as default for new tables

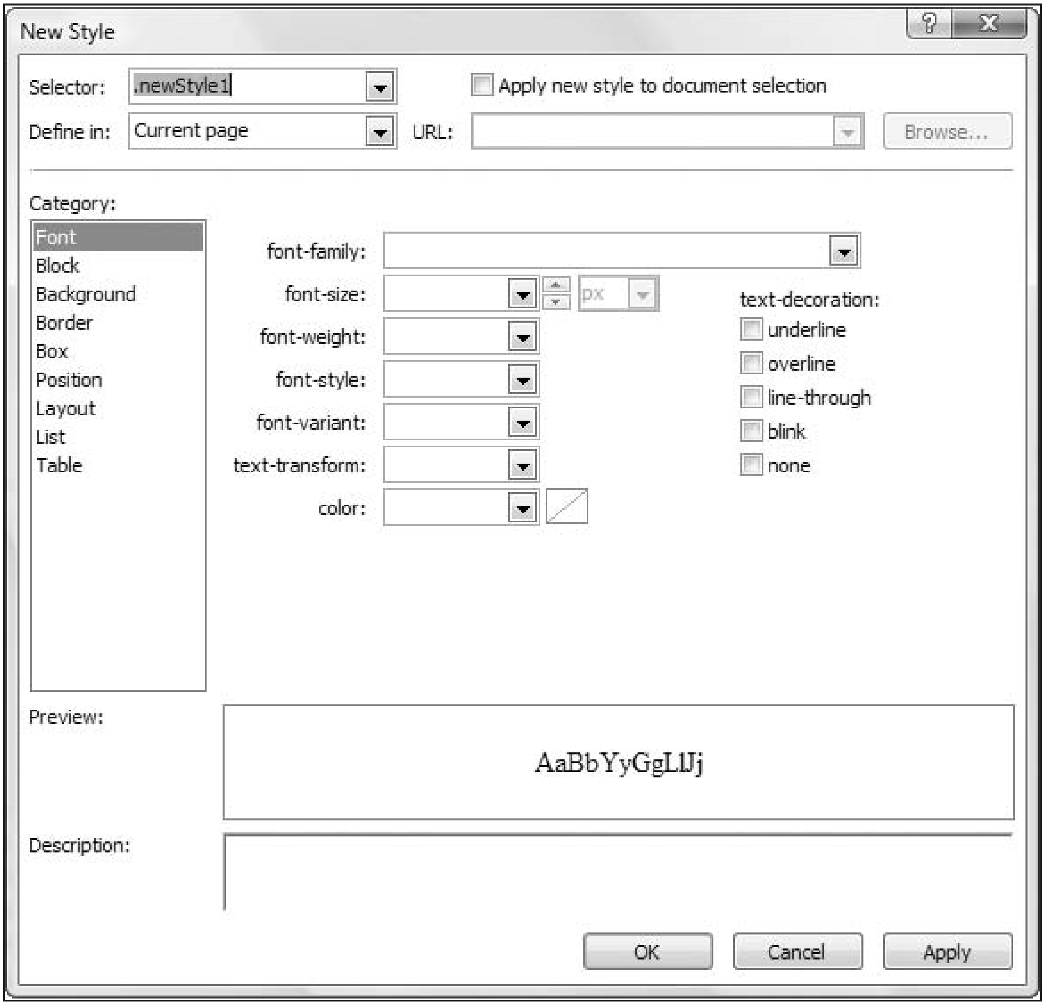
Figure 18-18

When you select an existing table in your Web page, the Table menu lets you insert and delete table rows, columns, and cells. The Modify menu option also allows you to split an existing cell into two separate cells, merge two cells into a single cell, and configure row and column sizing.

The Format menu includes basic element formatting options such as accessing the elements CSS class; setting fore and background colors, font, and position; and converting content to different types of lists.

Working with CSS in Visual Studio

Visual Studio 2008 offers a variety of new tools specifically designed to make working with CSS a much better experience. When working with the Visual Studio design surface, it's easy to create new styles for your Web page. You can either right-click on any object and select the New Style option from the content menu, or select the New Style option from the Format menu. Either option opens the New Style dialog box, shown in Figure 18-19.



Selector: .newStyle1

Define in: Current page

New Style
?
X
☐ Apply new style to document selection
URL:
Browse...
Category:
Font
Block
Background
Border
Box
Position
Layout
List
Table
font-family:
font-size:
px
font-weight:
font-style:
font-variant:
text-transform:
color:
text-decoration:
☐ underline
☐ overline
☐ line-through
☐ blink
☐ none
Preview:
AaBbYyGgLlJj
Description:
OK
Cancel
Apply

Figure 18-19

This dialog box makes creating a new style a snap. To start, select the type of Selector you want to create from the Selector drop-down list. The list includes all of the available element types, or if you want to create a Class or ID selector, simply type the Style name into the Selector combo box.

Next, you need to select where you want to create the style from the Define In combo box. You can select Current Page to create an internal style sheet, New Style Sheet to create a new external style sheet file, or

Existing Style Sheet to insert the style into an existing style sheet file. If you select either New Style Sheet or Existing Style Sheet, you will need to provide a value for the URL combo box.

Once you have entered the Selector you want to use and chosen a location to define the style, you can begin to set the styles properties. Simply select the property category from the Category list box and set the property values. The Preview area gives you a real-time preview of your new style. Additionally, the Description area shows you the actual property syntax created by Visual Studio. Click OK to close the dialog box.

After you begin to create styles for your application, you need to be able to manage and apply those styles to elements in your application. Visual Studio 2008 introduces three new tool windows you can use to manage style sheets, apply styles to elements, and easily inspect the style properties applied to an element.

Manage Styles Tool Window

The first tool to explore is the Manage Styles tool window, which can be opened by selecting Manage Styles from the CSS Styles submenu of the view menu. This tool window, shown in Figure 18-20, gives you the birds-eye view of all of the styles available to the current Web page open in Visual Studio.

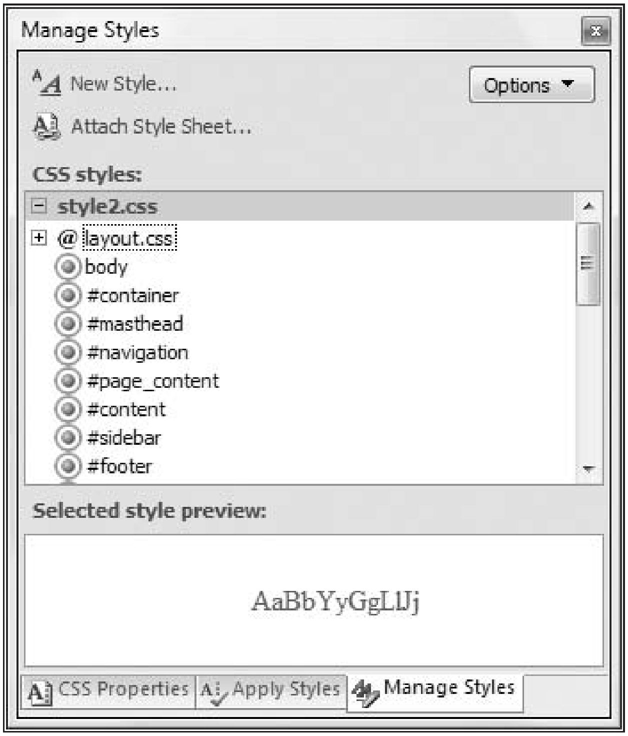

Manage Styles
New Style...
Options
Attach Style Sheet...
CSS styles:
style2.css
@ layout.css
body
#container
#masthead
#navigation
#page_content
#content
#sidebar
#footer
Selected style preview:
AaBbYyGgLlJj
CSS Properties
Apply Styles
Manage Styles

Figure 18-20

If you examine the contents of this tool, you see that the top portion includes two important links: New Style, which opens the New Style dialog box and allows you to create a new CSS styles as described earlier in this section, and the Attach Style Sheet link, which allows you to import new style sheets into a

Web page. Using this option to attach style sheets to the Web page causes Visual Studio to insert `<link>` tags into your Web page for you.

Remember that you need to be careful about the order of your link tags and style blocks in order to make sure that your styles are applied correctly.

The next portion of the tool window displays all of the styles available to the page. Styles are color coded according to their selector type using colored bullets: blue for Type Selectors, green for Class Selectors, and red for ID Selectors. Styles used within the page are shown with a gray circle surrounding the colored bullet. Should your Web page contain multiple linked style sheets, or inline styles sheets, these styles would be grouped together making it easy to determine where a style is defined.

Also, as you can see in Figure 18-20, the tool window also allows you to view style sheets attached to the Web page via the CSS Imports statement. By expanding the `layout.css` node in the figure, you can see a listing of all of the styles included in that style sheet.

Finally, the bottom of the tool window includes a preview area, allowing you to see a real-time preview of each style.

Apply Styles Tool Window

The second tool to help you use CSS in Visual Studio 2008 is the Apply Styles tool window. As with the Manage Style tool window, the Apply Styles tool window gives you a much easier way of viewing the CSS Styles available in your application and applying them to elements in a Web page. From the tool window, you can attach CSS files to the Web page, making external CSS Styles available; select page styles to apply or remove from an element; and modify styles. As with the other CSS tool windows, the Apply Styles tool window displays the available styles based on the CSS inheritance order, with external styles being shown first, then the page styles section, and finally the inline styles shown last. The Apply Styles also is contextually sensitive to the currently selected element and will show only those styles in your application that can be applied to the element type. Styles are also grouped according to the CSS Selector style, with a different visual indicator for each Selector type.

The tool window shown in Figure 18-21 shows the styles available for an anchor tag `<a>`. The tool first shows all styles in the attached `styles2.css` file, then the styles in the current page, and finally, if applied, the element's inline styles. You can click on styles in any of these sections to apply them to the element.

The Apply Styles tool also includes the intelligence to properly apply multiple class selectors (hold the Ctrl key down while you click class selectors in the list), but prevent you from selecting multiple ID selectors because that would result in invalid CSS. The tool will also not let you deselect type selectors or inline styles.

CSS Properties Tool Window

The final new tool is the CSS Properties tool window shown in Figure 18-22. This handy tool window shows you all of the CSS properties that have been applied to the currently selected element. The tool window is composed of two separate parts: the Applied Rules list and the CSS properties grid.

The Applied Rules list shows all of the CSS rules that are applied to the selected element. The list is automatically sorted to show you the inheritance chain of the applied rules with the outermost rules at

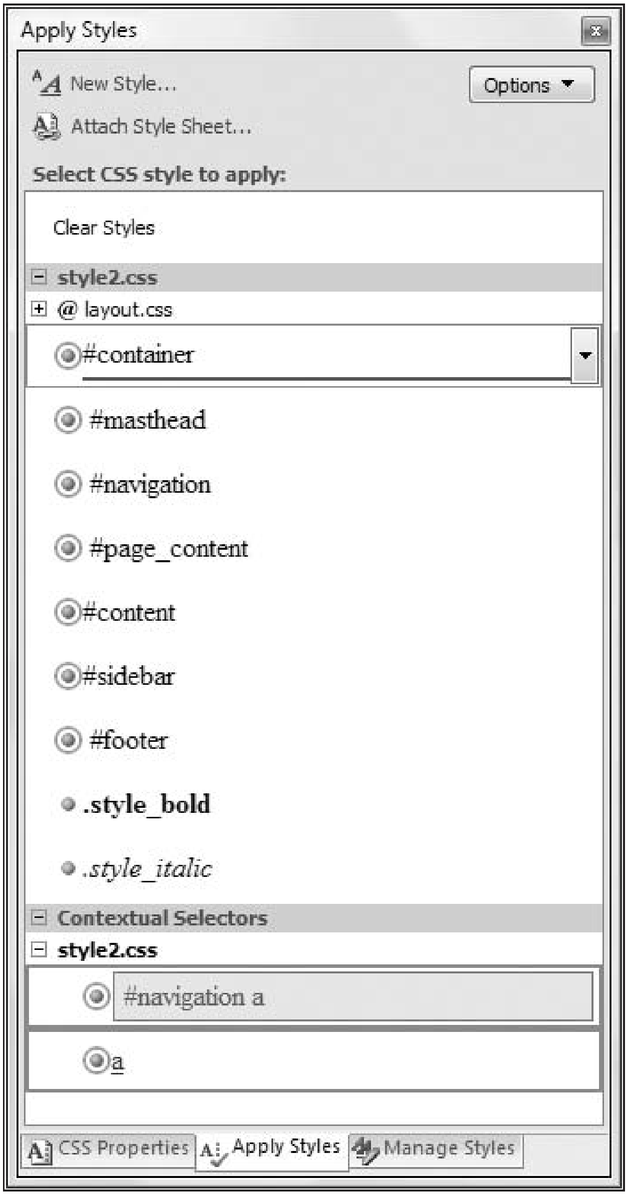
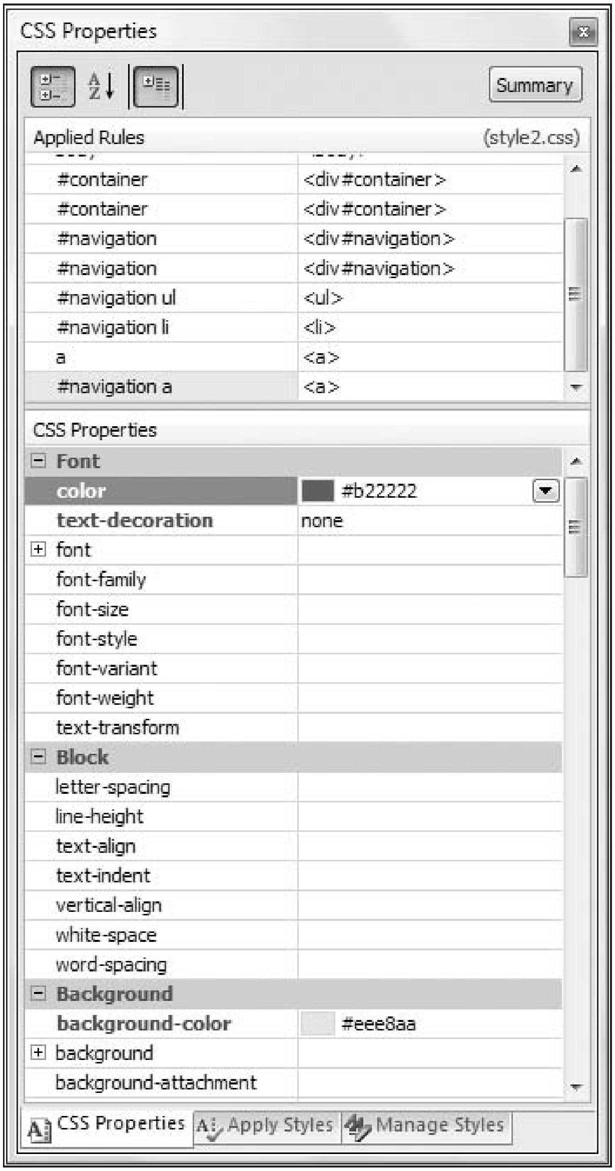

Apply Styles
New Style...
Attach Style Sheet...
Options
Select CSS style to apply:
Clear Styles
style2.css
@ layout.css
#container
#masthead
#navigation
#page_content
#content
#sidebar
#footer
.style_bold
.style_italic
Contextual Selectors
style2.css
#navigation a
a
CSS Properties
Apply Styles
Manage Styles

Figure 18-21

the top, moving down to the innermost rules. That means that rules contained in external CSS files are automatically sorted to the top of the list, and inline styles sorted to the bottom. You can click on each rule in the list and alter the properties that are shown in the CSS Properties grid displayed below.

The CSS Properties grid works in a similar fashion to the standard .NET properties grid, showing you all of the CSS properties available for the element, and properties that have values set being shown in



Applied Rules	(style2.css)
#container	<div#container>
#container	<div#container>
#navigation	<div#navigation>
#navigation	<div#navigation>
#navigation ul	
#navigation li	
a	<a>
#navigation a	<a>

CSS Properties	
⊟ **Font**	
color	■ #b22222
text-decoration	none
⊞ font	
font-family	
font-size	
font-style	
font-variant	
font-weight	
text-transform	
⊟ **Block**	
letter-spacing	
line-height	
text-align	
text-indent	
vertical-align	
white-space	
word-spacing	
⊟ **Background**	
background-color	□ #eee8aa
⊞ background	
background-attachment	

Figure 18-22

bold. Additionally, you can set property values for a CSS rule directly from the CSS property grid. Also in the CSS Properties tool window is a Summary button that allows you to change the display of the CSS Properties grid to show only properties that have values set. This can be very useful because HTML elements can have a large number of CSS properties.

Because CSS also includes the concept of property inheritance, which is generally not available in a standard .NET object, the CSS Rules list and CSS Properties grid have been designed to help you fully understand where a specific property value applied to an element is being defined. As you click on each rule in the CSS Rules list, the CSS Properties grid is updated to reflect that rule's properties. What you will notice, however, is that certain properties have a red strikethrough. (See Figure 18-23.)

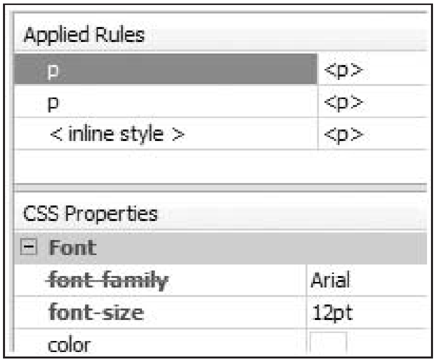


Figure 18-23

The strikethrough of the property indicates that the value of that property is being overridden by a rule closer to the element.

Managing Relative CSS Links in Masterpages

When working with CSS links in a masterpage, it can become difficult to manage the link to the CSS because the `Link` tag will not resolve a relative URL properly. A workaround for this is to define the tag as normal, but give it an ID. Then at runtime, in the Page Load event, simply set the `Href` of the link tag to the proper relative URL.

Styling ASP.NET Controls

Because ASP.NET controls simply render HTML markup, it is fairly easy to use CSS to style them. In fact, by default the controls actually already use inline CSS styles. You can see this in action by looking at the standard ASP.NET Button control. The standard method for styling ASP.NET controls like the Button is to provide values for the style related properties exposed by the control which is shown here:

```
<asp:Button ID="Button1" runat="server" BackColor="#3333FF"
    BorderColor="Silver" BorderStyle="Double" BorderWidth="3px"
    Font-Bold="True" Font-Size="Large" ForeColor="White" Text="Button" />
```

When ASP.NET processes the Web page containing this control, it converts a button into a standard HTML Input tag, and it also converts the style properties you have set into CSS styles and applies them to the Input tag. The HTML and CSS rendered by the button is shown here:

```
<input type="submit" name="Button1" value="Button" id="Button1"
    style="color:White;background-color:#3333FF;border-color:Silver;
border-width:3px;
            border-style:Double;font-size:Large;font-weight:bold;" />
```

Setting style properties directly on the ASP.NET controls is a fast and simple way to style the ASP.NET controls in your application. Additionally, because these are standard properties on the controls, you can also set them at runtime using code.

```
protected void Page_Load(object sender, EventArgs e)
{
    this.Button1.BackColor = System.Drawing.ColorTranslator.FromHtml("#3333FF");
    this.Button1.BorderColor = System.Drawing.Color.Silver;
    this.Button1.BorderStyle= BorderStyle.Double;
    this.Button1.BorderWidth = Unit.Pixel(3);
    this.Button1.Font.Bold=true;
    this.Button1.Font.Size=FontUnit.Large;
    this.Button1.ForeColor=System.Drawing.Color.White;
}
```

While using properties to set style info is easy and convenient, it does have some drawbacks, especially when you use the same technique with larger repeating controls. First, using inline styles makes it difficult to control the styling of your Web site at a higher, more abstract level. If you want every button in your Web site to have a specific style, generally you would have to manually set that style on every Button control in your entire site. Themes can help solve this but are not always useful, especially when you are mixing ASP.NET controls with standard HTML controls.

Second, for controls that generate a large amount of repetitive HTML, such as the GridView, having inline styles on every element of each iteration of the HTML adds a lot of extra weight to your Web page.

Thankfully, every ASP.NET server control exposes a `CssClass` property. This property allows you to provide one or more Class Selector Rules to the control. While this technique is helpful, and usually better than letting the control render inline styles, it still requires you to be familiar with the HTML that the control will render at runtime. Listing 18-13 shows how you can use the `CssClass` attribute to style the ASP.NET Button control.

Listing 18-13: Styling a standard ASP.NET Button control using CSS

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">

    <link href="SpringStyles.css" rel="stylesheet" type="text/css" />
    <style>
        .search
        {
            color:White;
            font-weight:bolder;
            background-color:Green;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Button ID="Button1" CssClass="search" runat="server" Text="Button" />
    </form>
</body>
</html>
```

In this case, the Button will have the search class applied to it.

ASP.NET 2.0 CSS–Friendly Control Adapters

If you are looking for more control over the rendering of the ASP.NET controls, especially so you can leverage more CSS-friendly development techniques, then you may want to look at the ASP.NET 2.0 CSS Friendly Control Adapters toolkit. This free download from Microsoft, available at `www.asp.net/cssadapters/`, leverages the ASP.NET Control Adapter model to allow you to have more control over how the in-box control render.

The toolkit includes CSS-friendly adapters for five of the standard ASP.NET controls: Menu, TreeView, DetailsView, FormView, and DataList. The CSS controls adapters for these controls modify the HTML they render so that it can be more easily styled with CSS. In fact, when these controls render using the CSS Friendly Control Adapters toolkit, they actually ignore most of the style-related properties that exist on the default control API.

To use CSS with these controls you simply create a style sheet using the same techniques that have been described easier in the chapter. Next you create CSS rules that map to the different parts of the control. The documentation included with the adapters gives you a great detailed view of the new control structure and how that maps to the CSS classes you can create.

Summary

CSS is a great way to add style to your Web site. It's a powerful and convenient mechanism that allows you to create complex styles and layouts for your Web site. A full discussion of CSS would require much more time and space than we have here, so this chapter focused on showing you some of the basic concepts of CSS, the tools Visual Studio provides you to work more easily with CSS, and how you can use CSS with ASP.NET server controls. If you are interested in learning more about the details of CSS, we recommended that you pick up the Wrox title *Beginning CSS: Cascading Style Sheets for Web Design, 2nd Edition* by Richard York (Wiley Publishing, Inc., 2007).

This chapter provided an overview of CSS, introducing you to external, internal, and inline style sheets. You learned about the various Selector types the CSS offers and about basic layout and positioning of CSS elements, including how the box model works to influence element positions in your Web page.

Next you reviewed the tools available in Visual Studio that make working with CSS easy. New tools in Visual Studio 2008, including the Style Manager and CSS properties tool windows, make working with CSS easier than ever.

Finally, you looked at how you can use CSS with the ASP.NET server controls, and how you can use the ASP.NET 2.0 CSS Friendly Control Adapters to have more control over the style of the standard ASP.NET server controls.

19

ASP.NET AJAX

AJAX is definitely the hot buzzword in the Web application world at the moment. AJAX is an acronym for *Asynchronous JavaScript and XML* and, in Web application development, it signifies the capability to build applications that make use of the `XMLHttpRequest` object.

The creation and the inclusion of the `XMLHttpRequest` object in JavaScript and the fact that most upper-level browsers support it led to the creation of the AJAX model. AJAX applications, although they have been around for a few years, gained popularity after Google released a number of notable, AJAX-enabled applications such as Google Maps and Google Suggest. These applications demonstrated the value of AJAX.

Shortly thereafter, Microsoft released a beta for a new toolkit that enabled developers to incorporate AJAX features in their Web applications. This toolkit, code-named *Atlas* and later renamed ASP.NET AJAX, makes it extremely simple to start using AJAX features in your applications today.

The ASP.NET AJAX toolkit was *not* part of the default .NET Framework 2.0 install. If you are using the .NET Framework 2.0, it is an extra component that you must download from the Internet. If you are using ASP.NET 3.5, you don't have to worry about installing the ASP.NET AJAX toolkit as everything you need is already in place for you.

Understanding the Need for AJAX

Today, if you are going to build an application, you have the option of creating a thick-client or a thin-client application. A *thick-client* application is typically a compiled executable that end users can run in the confines of their own environment — usually without any dependencies elsewhere (such as an upstream server). Generally, the technology to build this type of application is the Windows Forms technology, or MFC in the C++ world. A *thin-client* application is typically one that has its processing and rendering controlled at a single point (the upstream server) and the results of the view are sent down as HTML to a browser to be viewed by a client. To work, this type of technology generally requires that the end user be connected to the Internet or an intranet of some kind.

Each type of application has its pros and cons. The thick-client style of application is touted as more fluid and more responsive to an end user's actions. In a Web-based application, the complaint has been for many years that every action by an end user takes numerous seconds and results in a jerky page refresh. In turn, the problem with a thick-client style of application has always been that the application sits on the end user's machine and any patches or updates to the application require you to somehow upgrade each and every machine upon which the application sits. In contrast, the thin-client application, or the Web application architecture includes only one instance of the application. The application in this case is installed on a Web server and any updates that need to occur happen only to this instance. End users who are calling the application through their browsers always get the latest and greatest version of the application. That change model has a lot of power to it.

With this said, it is important to understand that Microsoft is making huge inroads into solving this thick- or thin-client problem and you now have options that completely change this model. For instance, the Windows Presentation Foundation technology recently offered by Microsoft and the new Silverlight technology blur the lines between to two traditional application styles.

Even with the existing Windows Forms and ASP.NET technologies to build the respective thick- or thin-client applications, each of these technologies are advancing to a point where they are even blurring the lines further. ASP.NET AJAX in particular is further removing any of the negatives that would have stopped you from building an application on the Web.

ASP.NET AJAX makes your Web applications seem more fluid than ever before. AJAX-enabled applications are responsive and give the end user immediate feedback and direction through the workflows that you provide. The power of this alone makes the study of this new technology and its incorporation into your projects of the utmost importance.

Before AJAX

So, what is AJAX doing to your Web application? First off, let's take a look at what a Web page does when it *does not* use AJAX. Figure 19-1 shows a typical request and response activity for a Web application.

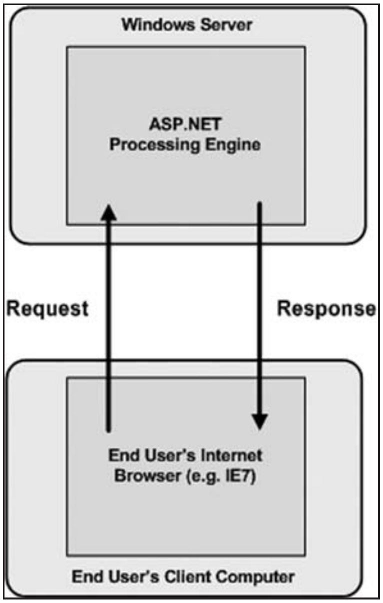

Windows Server
ASP.NET Processing Engine
Request
Response
End User's Internet Browser (e.g. IE7)
End User's Client Computer

Figure 19-1

In this case, an end user makes a request from his browser to the application that is stored on your Web server. The server processes the request and ASP.NET renders a page, which is then sent to the requestor as a response. The response, once received by the end user, is displayed within the end user's browser.

From here, many events that take place within the application instance as it sits within the end user's browser causes the complete request and response process to reoccur. For instance, the end user might click a radio button, a check box, a button, a calendar, or anything else and this causes the entire Web page to be refreshed or a new page to be provided.

AJAX Changes the Story

On the other hand, an AJAX-enabled Web page includes a JavaScript library on the client that takes care of issuing the calls to the Web server. It does this when it is possible to send a request and get a response for just part of the page and using script; the client library updates that part of the page without updating the entire page. An entire page is a lot of code to send down to the browser to process each and every time. By just processing part of the page, the end user experiences what some people term ''fluidity'' in the page, which makes the page seem more responsive. The amount of code required to update just a portion of a page is less and produces the responsiveness the end user expects. Figure 19-2 shows a diagram of how this works.

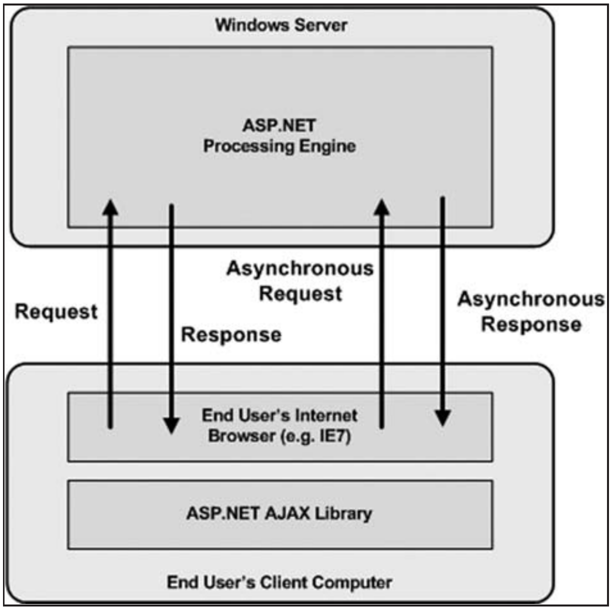


Figure 19-2

Figure 19-2, demonstrates that the first thing that happens is the entire page is delivered in the initial request and response. From there, any partial updates required by the page are done using the client script library. This library can make asynchronous page requests and update just the portion of the page that needs updating. One major advantage to this is that a minimal amount of data is transferred for the updates to occur. Updating a partial page is better than recalling the entire page for what is just a small change to the page.

AJAX is dependent upon a few technologies in order for it to work. The first is the `XMLHttpRequest` object. This object allows the browser to communicate to a back-end server and has been available in the Microsoft world since Internet Explorer 5 through the MSXML ActiveX component. Of course, the other major component is JavaScript. This technology provides the client-side initiation to communication with the back-end services and takes care of packaging a message to send to any server-side services. Another aspect of AJAX is support for DHTML and the Document Object Model (DOM). These are the pieces that will change the page when the asynchronous response is received from the server. Finally, the last piece is the data that is being transferred from the client to the server. This is done in XML or, more important, JSON.

Support for the `XMLHttpRequest` object gives JavaScript functions within the client script library the capability to call server-side events. As stated, typically HTTP requests are issued by a browser. It is also the browser that takes care of processing the response that comes from the server and then usually regenerates the entire Web page in the browser after a response is issued. This process is detailed in Figure 19-3.

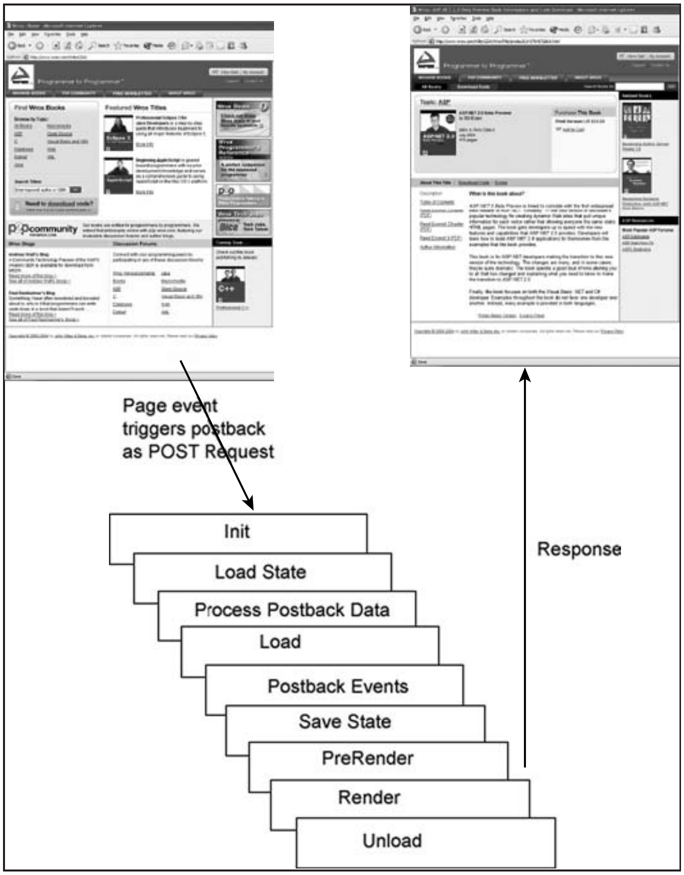

Page event
triggers postback
as POST Request
Init
Load State
Process Postback Data
Load
Postback Events
Save State
PreRender
Render
Unload
Response

Figure 19-3

If you use the `XMLHttpRequest` object from your JavaScript library, you do not actually issue full page requests from the browser. Instead, you use a client-side script engine (which is basically a JavaScript function) to initiate the request and also to receive the response. Because you are also not issuing a request and response to deal with the entire Web page, you can skip a lot of the page processing because it is not needed. This is the essence of an AJAX Web request. It is illustrated in Figure 19-4.

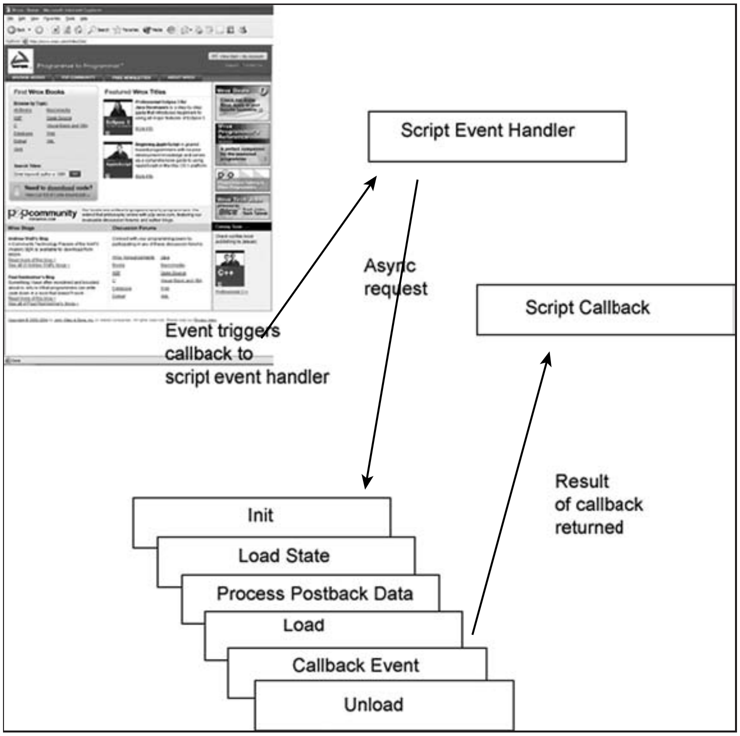

Script Event Handler
Async request
Script Callback
Event triggers callback to script event handler
Result of callback returned
Init
Load State
Process Postback Data
Load
Callback Event
Unload

Figure 19-4

As stated, this opens the door to a tremendous number of possibilities. Microsoft has provided the necessary script engines to automate much of the communication that must take place in order for AJAX-style functionality to occur.

ASP.NET AJAX and Visual Studio 2008

Prior to Visual Studio 2008, the ASP.NET AJAX product used to be a separate installation that you were required to install on your machine and the Web server that you were working with. This release gained in popularity quite rapidly and is now a part of the Visual Studio 2008 offering. Not only is it a part of the Visual Studio 2008 IDE, the ASP.NET AJAX product is also baked into the .NET Framework 3.5. This means that to use ASP.NET AJAX, you don't need to install anything if you are working with ASP.NET 3.5.

> *If you are using an ASP.NET version that is prior to the ASP.NET 3.5 release, then you need to visit* `www.asp.net/AJAX` *to get the components required to work with AJAX.*

ASP.NET AJAX is now just part of the ASP.NET framework. When you create a new Web application, you do not have to create a separate type of ASP.NET application. Instead, all ASP.NET applications that you create are now AJAX-enabled.

If you have already worked with ASP.NET AJAX prior to this 3.5 release, you will find that there is really nothing new to learn. The entire technology is seamlessly integrated into the overall development experience.

Overall, Microsoft has fully integrated the entire ASP.NET AJAX experience so you can easily use Visual Studio and its visual designers to work with your AJAX-enabled pages and even have the full debugging story that you would want to have with your applications. Using Visual Studio 2008, you are now able to debug the JavaScript that you are using in the pages.

In addition, it is important to note that Microsoft focused a lot of attention on cross-platform compatibility with ASP.NET AJAX. You will find that the AJAX-enabled applications that you build upon the .NET Framework 3.5 can work within all the major up-level browsers out there (e.g., Firefox and Opera).

Client-Side Technologies

There really are two parts of the ASP.NET AJAX story. The first is a client-side framework and a set of services that are completely on the client-side. The other part of the story is a server-side framework. Remember that the client-side of ASP.NET AJAX is all about the client communicating asynchronous requests to the server-side of the offering.

For this reason, Microsoft offers a Client Script Library, which is a JavaScript library that takes care of the required communications. The Client Script Library is presented in Figure 19-5.

The Client Script Library provides a JavaScript, object-oriented interface that is reasonably consistent with aspects of the .NET Framework. Because browser compatibility components are built in, any work that you build in this layer or (in most cases) work that you let ASP.NET AJAX perform for you here will function with a multitude of different browsers. Also, several components support a rich UI infrastructure that produces many things that would take some serious time to build yourself.

The interesting thing about the client-side technologies that are provided by ASP.NET AJAX is that they are completely independent of ASP.NET. In fact, any developer can freely download the Microsoft AJAX Library (again from `asp.net/AJAX`) and use it with other Web technologies such as PHP (`php.net`) and Java Server Pages (JSP). With that said, really the entire Web story is a lot more complete with the server-side technologies that are provided with ASP.NET AJAX.

Server-Side Technologies

As an ASP.NET developer, you will most likely be spending most of your time on the server-side aspect of ASP.NET AJAX. Remember that ASP.NET AJAX is all about the client-side technologies talking back to the server-side technologies. You can actually perform quite a bit on the server-side of ASP.NET AJAX.

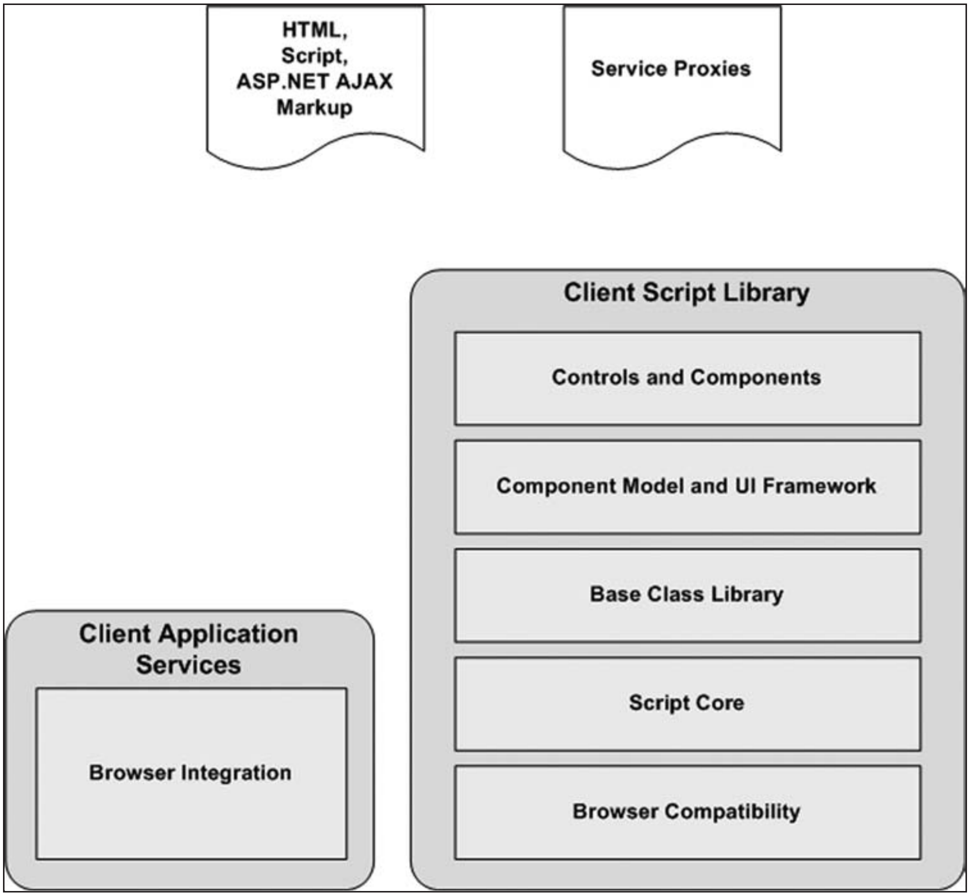

HTML,
Script,
ASP.NET AJAX
Markup
Service Proxies
Client Script Library
Controls and Components
Component Model and UI Framework
Base Class Library
Script Core
Browser Compatibility
Client Application Services
Browser Integration

Figure 19-5

The server-side framework knows how to deal with client requests (e.g., putting responses in the correct format). The server-side framework also takes care of the marshalling of objects back and forth between JavaScript objects and the .NET objects that you are using in your server-side code. Figure 19-6 illustrates the server-side framework provided by ASP.NET AJAX.

When you have the .NET Framework 3.5, you will have the ASP.NET AJAX Server Extensions on top of the core ASP.NET 2.0 Framework, the Windows Communication Foundation, as well as ASP.NET-based Web services (`.asmx`).

Developing with ASP.NET AJAX

There are a couple of types of Web developers out there. There are the Web developers who are used to working with ASP.NET and who have experience working with server-side controls and manipulating these controls on the server-side. Then there are developers who concentrate on the client-side and work with DHTML and JavaScript to manipulate and control the page and its behaviors.

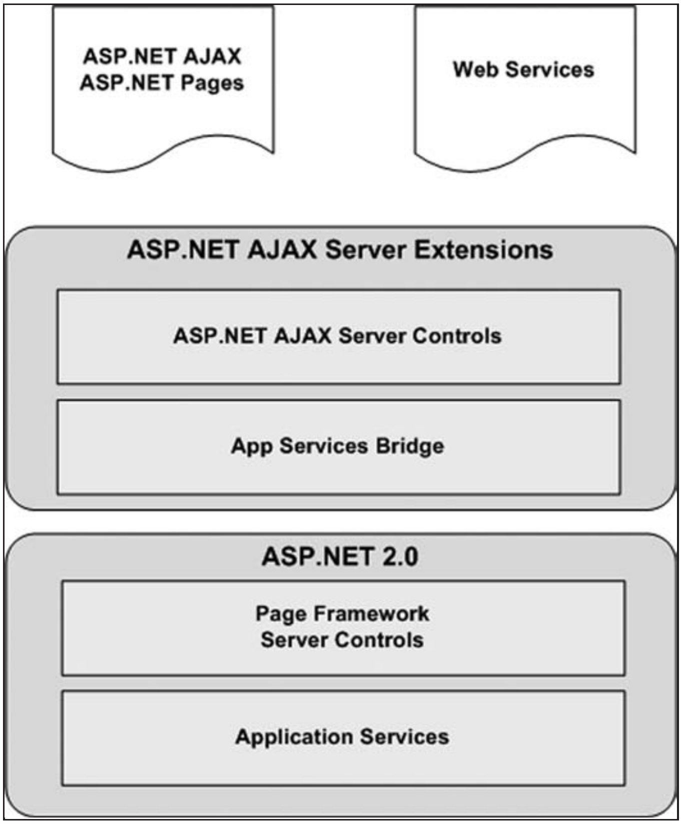

ASP.NET AJAX
ASP.NET Pages
Web Services
ASP.NET AJAX Server Extensions
ASP.NET AJAX Server Controls
App Services Bridge
ASP.NET 2.0
Page Framework
Server Controls
Application Services

Figure 19-6

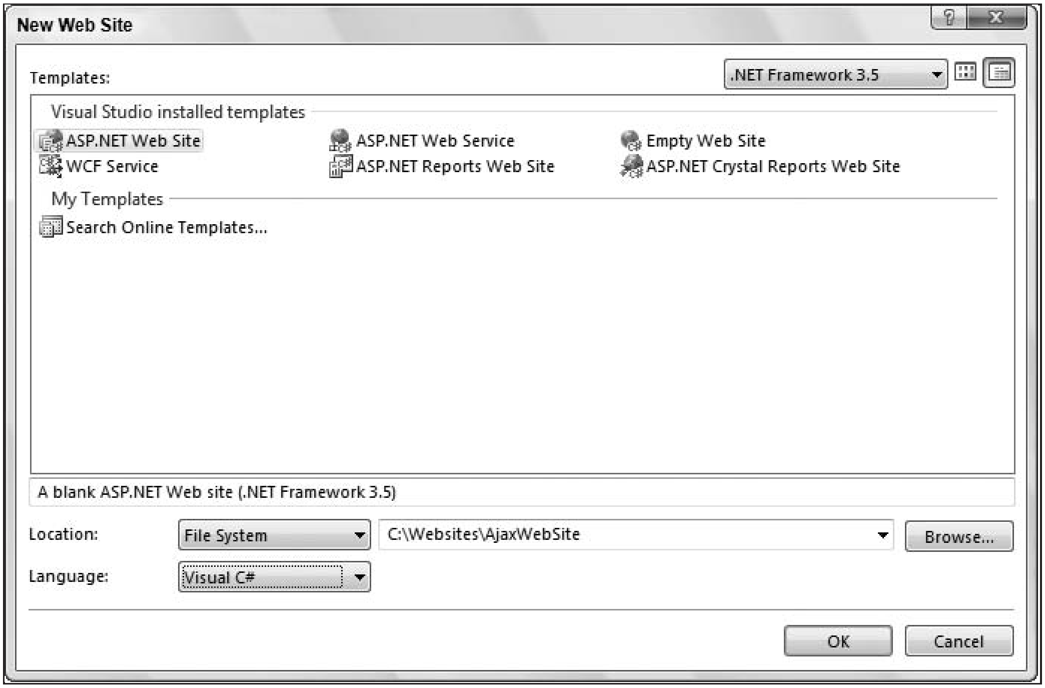
With that said, it is important to realize that ASP.NET AJAX was designed for both types of developers. If you want to work more on the server-side of ASP.NET AJAX, you can use the new ScriptManager control and the new UpdatePanel control to AJAX-enable your current ASP.NET applications with little work on your part. All this work can be done using the same programming models that you are quite familiar with in ASP.NET.

Both the ScriptManager and the UpdatePanel controls are discussed later in this chapter.

In turn, you can also use the Client Script Library directly and gain greater control over what is happening on the client's machine. Next, this chapter looks at building a simple Web application that makes use of AJAX.

ASP.NET AJAX Applications

The next step is to build a basic sample utilizing this new framework. First create a new ASP.NET Web Site application using the New Web Site dialog. Name the project `AJAXWebSite`. You will notice (as shown here in Figure 19-7) that there is not a separate type of ASP.NET project for building an ASP.NET AJAX application because every ASP.NET application that you now build is AJAX-enabled.


New Web Site
Templates:
.NET Framework 3.5
Visual Studio installed templates
ASP.NET Web Site
ASP.NET Web Service
Empty Web Site
WCF Service
ASP.NET Reports Web Site
ASP.NET Crystal Reports Web Site
My Templates
Search Online Templates...
Browse...
OK
Cancel

A blank ASP.NET Web site (.NET Framework 3.5)

Location: File System C:\Websites\AjaxWebSite

Language: Visual C#

Figure 19-7

After you create the application, you will be presented with what is now a standard Web Site project. However, you may notice some additional settings in the `web.config` file that are new to the ASP.NET 3.5. At the top of the `web.config` file, there are new configuration sections that are registered that deal with AJAX. This section of `web.config` is presented in Listing 19-1.

Listing 19-1: The <configSections> element for an ASP.NET 3.5 application

```
<?xml version="1.0"?>

<configuration>
  <configSections>
     <sectionGroup name="system.web.extensions"
       type="System.Web.Configuration.SystemWebExtensionsSectionGroup,
        System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
        PublicKeyToken=31BF3856AD364E35">
        <sectionGroup name="scripting"
          type="System.Web.Configuration.ScriptingSectionGroup,
            System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
            PublicKeyToken=31BF3856AD364E35">
          <section name="scriptResourceHandler"
           type="System.Web.Configuration.ScriptingScriptResourceHandlerSection,
              System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
              PublicKeyToken=31BF3856AD364E35"
            requirePermission="false"
           allowDefinition="MachineToApplication"/>
```

Continued

```
            <sectionGroup name="webServices"
             type="System.Web.Configuration.ScriptingWebServicesSectionGroup,
               System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
               PublicKeyToken=31BF3856AD364E35">
               <section name="jsonSerialization"
                type="System.Web.Configuration.ScriptingJsonSerializationSection,
                   System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
                   PublicKeyToken=31BF3856AD364E35" requirePermission="false"
                allowDefinition="Everywhere" />
               <section name="profileService"
                type="System.Web.Configuration.ScriptingProfileServiceSection,
                   System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
                   PublicKeyToken=31BF3856AD364E35" requirePermission="false"
                allowDefinition="MachineToApplication" />
              <section name="authenticationService"
               type="System.Web.Configuration.ScriptingAuthenticationServiceSection,
                 System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
                 PublicKeyToken=31BF3856AD364E35" requirePermission="false"
              allowDefinition="MachineToApplication" />
              <section name="roleService"
               type="System.Web.Configuration.ScriptingRoleServiceSection,
                  System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
                  PublicKeyToken=31BF3856AD364E35" requirePermission="false"
               allowDefinition="MachineToApplication" />
            </sectionGroup>
         </sectionGroup>
      </sectionGroup>
   </configSections>

<!-- Configuraiton removed for clarity -->

</configuration>
```

With the `web.config` file in place (as provided by the ASP.NET Web Site project type), the next step is to build a simple ASP.NET page that does not yet make use of AJAX.

Building a Simple ASP.NET Page Without AJAX

The first step is to build a simple page that does not yet make use of the AJAX capabilities offered by ASP.NET 3.5. Your page needs only a Label control and Button server control. The code for the page is presented in Listing 19-2.

Listing 19-2: A simple ASP.NET 3.5 page that does not use AJAX

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Label1.Text = DateTime.Now.ToString()
    End Sub</script>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>My Normal ASP.NET Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server"></asp:Label>
        <br />
        <br />
        <asp:Button ID="Button1" runat="server" Text="Click to get machine time"
            onclick="Button1_Click" />
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = DateTime.Now.ToString();
    }
</script>
```

When you pull this page up in the browser, it contains only a single button. When the button is clicked, the Label control that is on the page is populated with the time from the server machine. Before the button is clicked, the page's code is similar to the code presented in Listing 19-3.

Listing 19-3: The page output for a page that is not using AJAX

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
 My Normal ASP.NET Page
</title></head>
<body>
    <form name="form1" method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
 value="/wEPDwULLTE4OTg4OTc0MjVkZIgwrMMmvqXJHfogxzgZ92wTUORS" />
</div>
    <div>
        <span id="Label1"></span>
        <br />
        <br />
        <input type="submit" name="Button1" value="Click to get machine time"
         id="Button1" />
    </div>
```

Continued

```
<div>

 <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
  value="/wEWAgLFpoapCAKM54rGBkhUDe2q/7eVsROfd9QCMK6CwiI7" />
</div></form>
</body>
</html>
```

There is not much in this code. There is a little ViewState and a typical form that will be posted back to the `Default.aspx` page. When the end user clicks the button on the page, a full post back to the server occurs and the entire page is reprocessed and returned to the client's browser. Really, the only change made to the page is that the `<span id="Lable1"></span>` element is populated with a value, but in this case, the entire page is returned.

Building a Simple ASP.NET Page with AJAX

The next step is to build upon the page from Listing 19-2 and add AJAX capabilities to it. For this example, you will be adding some additional controls. Two of the controls to add are typical ASP.NET server controls — another Label and Button server control. In addition to these controls, you are going to have to add some ASP.NET AJAX controls.

In the Visual Studio 2008 toolbox, you will find a new section titled AJAX Extensions. This new section is shown in Figure 19-8.

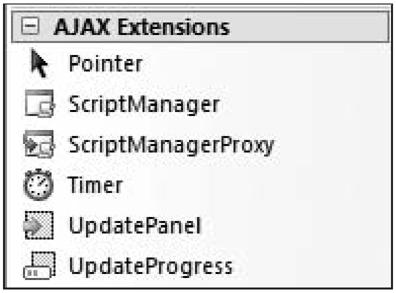


Figure 19-8

From AJAX Extensions, add a ScriptManager server control to the top of the page and include the second Label and Button control inside the UpdatePanel control. The UpdatePanel control is a template server control and allows you to include any number of items within it (just as other templated ASP.NET server controls). When you have your page set up, it should look something like Figure 19-9.

The code for this page is shown in Listing 19-4.

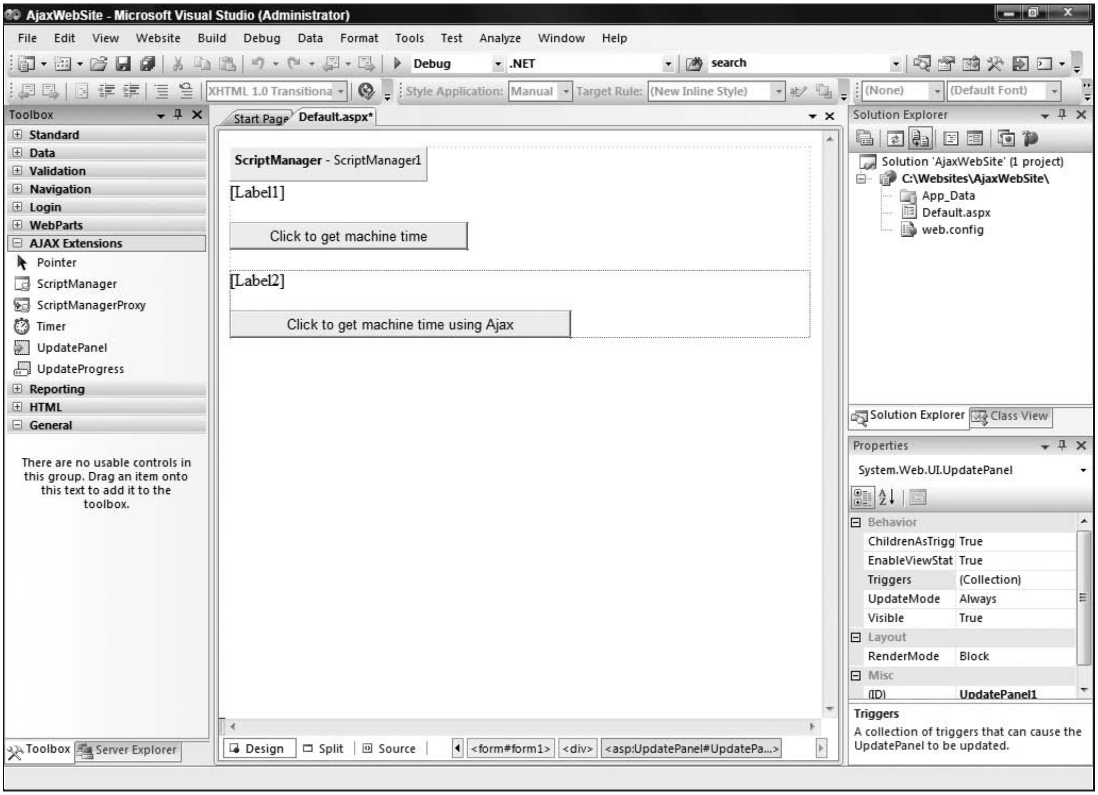
Listing 19-4: A simple ASP.NET AJAX page

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)
       Label1.Text = DateTime.Now.ToString()
    End Sub

    Protected Sub Button2_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)
       Label2.Text = DateTime.Now.ToString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>My ASP.NET AJAX Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <asp:Label ID="Label1" runat="server"></asp:Label>
        <br />
        <br />
        <asp:Button ID="Button1" runat="server" Text="Click to get machine time"
            onclick="Button1_Click" />
        <br />
        <br />
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <asp:Label ID="Label2" runat="server" Text=""></asp:Label>
                <br />
                <br />
                <asp:Button ID="Button2" runat="server"
                 Text="Click to get machine time using AJAX"
                 onclick="Button2_Click" />
            </ContentTemplate>
        </asp:UpdatePanel>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = DateTime.Now.ToString();
    }

    protected void Button2_Click(object sender, EventArgs e)
    {
        Label2.Text = DateTime.Now.ToString();
    }
</script>
```


AjaxWebSite - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Data Format Tools Test Analyze Window Help
Debug
.NET
search
XHTML 1.0 Transitiona
Style Application: Manual
Target Rule: (New Inline Style)
(None)
(Default Font)
Toolbox
Standard
Data
Validation
Navigation
Login
WebParts
AJAX Extensions
Pointer
ScriptManager
ScriptManagerProxy
Timer
UpdatePanel
UpdateProgress
Reporting
HTML
General
Toolbox
Server Explorer
Start Page
Default.aspx*
ScriptManager - ScriptManager1
[Label1]
Click to get machine time
[Label2]
Click to get machine time using Ajax
Design
Split
Source
<form#form1>
<div>
<asp:UpdatePanel#UpdatePa...>
Solution Explorer
Solution 'AjaxWebSite' (1 project)
C:\Websites\AjaxWebSite\
App_Data
Default.aspx
web.config
Solution Explorer
Class View
Properties
System.Web.UI.UpdatePanel
Behavior
Layout
Misc

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

ChildrenAsTrigg True
EnableViewStat True
Triggers (Collection)
UpdateMode Always
Visible True
RenderMode Block
(ID) UpdatePanel1

Triggers
A collection of triggers that can cause the UpdatePanel to be updated.

Figure 19-9

When this page is pulled up in the browser, it has two buttons. The first button causes a complete page postback and updates the current time in the Label1 server control. Clicking on the second button causes an AJAX asynchronous postback. Clicking this second button updates the current server time in the Label2 server control. When you click the AJAX button, the time in Label1 will not change at all, as it is outside of the UpdatePanel. A screenshot of the final result is presented in Figure 19-10.

When you first pull up the page from Listing 19-4, the code of the page is quite different from the page that was built without using AJAX. Listing 19-5 shows the page results that you will see.

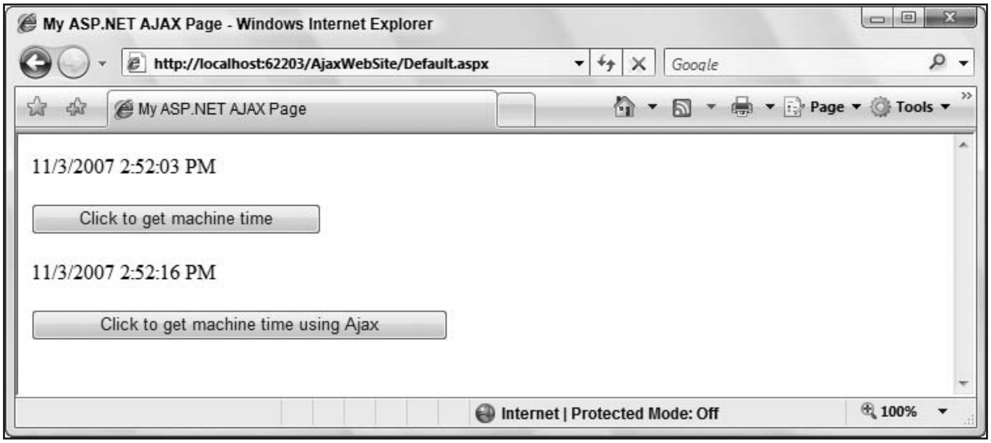
Listing 19-5: The page output for a page that is using AJAX

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
My ASP.NET AJAX Page
</title></head>
<body>
    <form name="form1" method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value="" />
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
 value="/wEPDwULLTE4NzE5NTc5MzRkZDRIzHpPZg4GaO9Hox9A/RnOflkm" />
```

```
</div>

<script type="text/javascript">
//<![CDATA[
var theForm = document.forms['form1'];
if (!theForm) {
    theForm = document.form1;
}
function __doPostBack(eventTarget, eventArgument) {
    if (!theForm.onsubmit || (theForm.onsubmit() != false)) {
        theForm.__EVENTTARGET.value = eventTarget;
        theForm.__EVENTARGUMENT.value = eventArgument;
        theForm.submit();
    }
}
//]]>
</script>

<script src="/AJAXWebSite/WebResource.axd?d=o84znEj-
 n4cYi0Wg0pFXCg2&amp;t=633285028458684000" type="text/javascript"></script>

<script src="/AJAXWebSite/ScriptResource.axd?
 d=FETsh5584DXpx8XqIhEM50YSKyR2GkoMoAqraYEDU5_
 gi1SUmL2Gt7rQTRBAw56lSojJRQe0OjVI8SiYDjmpYmFP0
 CO8wBFGhtKKJwm2MeE1&amp;t=633285035850304000" type="text/javascript"></script>
<script type="text/javascript">
//<![CDATA[
if (typeof(Sys) === 'undefined') throw new Error('ASP.NET AJAX client-side
  framework failed to load.');
//]]>
</script>

<script src="/AJAXWebSite/ScriptResource.axd?
 d=FETsh5584DXpx8XqIhEM50YSKyR2GkoMoAqraYEDU5_
 gi1SUmL2Gt7rQTRBAw56l7AYfmRViCoO2lZ3XwZ33TGiC
 t92e_UOqfrP30mdEYnJYs09ulU1xBLj8TjXOLR1k0&amp;t=633285035850304000"
 type="text/javascript"></script>
    <div>
        <script type="text/javascript">
//<![CDATA[
Sys.WebForms.PageRequestManager._initialize('ScriptManager1',
document.getElementById('form1'));
Sys.WebForms.PageRequestManager.getInstance()._updateControls(['tUpdatePanel1'],
[], [], 90);
//]]>
</script>
        <span id="Label1"></span>
        <br />
        <br />
        <input type="submit" name="Button1" value="Click to get machine time"
         id="Button1" />
        <br />
        <br />
        <div id="UpdatePanel1">
```

Continued

```
                <span id="Label2"></span>
                <br />
                <br />
                <input type="submit" name="Button2" value="Click to get machine
                 time using AJAX" id="Button2" />
</div>
    </div>
<div>

 <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
value="/wEWAwLktbDGDgKM54rGBgK7q7GGCMYnNq57VIqmVD2sRDQqfnOsgWQK" />
</div>

<script type="text/javascript">
//<![CDATA[
Sys.Application.initialize();
//]]>
</script>
</form>
</body>
</html>
```


My ASP.NET AJAX Page - Windows Internet Explorer
http://localhost:62203/AjaxWebSite/Default.aspx
Google
My ASP.NET AJAX Page
Page
Tools
Click to get machine time
Click to get machine time using Ajax
Internet | Protected Mode: Off
100%

11/3/2007 2:52:03 PM

11/3/2007 2:52:16 PM

Figure 19-10

From there, if you click Button1 and perform the full-page postback, you get this entire bit of code back in a response — even though you are interested in updating only a small portion of the page! However, if you click Button2 — the AJAX button — you send the request shown in Listing 19-6.

Listing 19-6: The asynchronous request from the ASP.NET AJAX page

```
POST /AJAXWebSite/Default.aspx HTTP/1.1
Accept: */*
Accept-Language: en-US
Referer: http://localhost.:62203/AJAXWebSite/Default.aspx
x-microsoftAJAX: Delta=true
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Cache-Control: no-cache
```

```
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1;
.NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 1.1.4322; .NET CLR
3.5.21004; .NET CLR 3.0.04506)
Host: localhost.:62203
Content-Length: 334
Proxy-Connection: Keep-Alive
Pragma: no-cache

ScriptManager1=UpdatePanel1%7CButton2&__EVENTTARGET=&__EVENTARGUMENT=&__VIEWSTATE=%
2FwEPDwULLTE4NzE5NTc5MzQPZBYCAgQPZBYCAgMPDxYCHgRUZXh0BRQxMS8zLzIwMDcgMjoxNzo1NSBQTW
RkZHZxUyYQG0M25t8U7vLbHRJuKlcS&__EVENTVALIDATION=%2FwEWAwKCxdk9AoznisYGArursYYI1844
hk7V466AsW31G5yIZ73%2Bc6o%3D&Button2=Click%20to%20get%20machine%20time%20
using%20Aj
ax
```

The response for this request is shown in Listing 19-7:

Listing 19-7: The asynchronous response from the ASP.NET AJAX page

```
HTTP/1.1 200 OK
Server: ASP.NET Development Server/9.0.0.0
Date: Sat, 03 Nov 2007 19:17:58 GMT
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/plain; charset=utf-8
Content-Length: 796
Connection: Close

239|updatePanel|UpdatePanel1|
                <span id="Label2">11/3/2007 2:17:58 PM</span>
                <br />
                <br />
                <input type="submit" name="Button2" value="Click to get machine
                 time using AJAX" id="Button2" />
|172|hiddenField|__VIEWSTATE|/wEPDwULLTE4NzE5NTc5MzQPZBYCAgQPZBYEAgMPDxYCHgRUZXh0BR
QxMS8zLzIwMDcgMjoxNzo1NSBQTWRkAgcPZBYCZg9kFgICAQ8PFgIfAAUUMTEvMy8yMDA3IDI6MTc6NTggU
E1kZGQ4ipZIg91+XSI/dqxFueSUwcrXGw==|56|hiddenField|__EVENTVALIDATION|/wEWAwKCz4mbCA
K7q7GGCAKM54rGBj8b4/mkKNKhV59qX9SdCzqU3AiM|0|asyncPostBackControlIDs|||0|postBackCo
ntrolIDs|||13|updatePanelIDs||tUpdatePanel1|0|childUpdatePanelIDs|||12|panelsToRefr
eshIDs||UpdatePanel1|2|asyncPostBackTimeout||90|12|formAction||Default.aspx|22|page
Title||My ASP.NET AJAX Page|
```

From Listing 19-7 here, you can see that the response is much smaller than an entire Web page! In fact, the main part of the response is only the code that is contained within the UpdatePanel server control and nothing more. The items at the bottom deal with the ViewState of the page (as it has now changed) and some other small page changes.

ASP.NET AJAX's Server-Side Controls

When you look at the AJAX Extensions section in the Visual Studio 2008 toolbox, you will notice that there are not many controls there at your disposal. The controls there are focused on allowing you to

AJAX-enable your ASP.NET applications. They are enabling controls. If you are looking for more specific server controls that take advantage of the new AJAX model, then look at the ASP.NET AJAX Control Toolkit — a separate download that is covered in the next chapter.

The new ASP.NET AJAX server controls that come with ASP.NET 3.5 are discussed in the following table.

ASP.NET AJAX Server Control	Description
ScriptManager	A component control that manages the marshalling of messages to the AJAX-enabled server for the parts of the page requiring partial updates. Every ASP.NET page will require a ScriptManager control in order to work. It is important to note that you can have only a single ScriptManager control on a page.
ScriptManagerProxy	A component control that acts as a ScriptManager control for a content page. The ScriptManagerProxy control, which sits on the content page (or sub-page), works in conjunction with a required ScriptManager control that resides on the master page.
Timer	The Timer control will execute client-side events at specific intervals and allows specific parts of your page to update or refresh at these moments.
UpdatePanel	A container control that allows you to define specific areas of the page that are enabled to work with the ScriptManager. These areas can then, in turn, make the partial page postbacks and update themselves outside the normal ASP.NET page postback process.
UpdateProgress	A control that allows you to display a visual element to the end user to show that a partial-page postback is occurring to the part of the page making the update. This is an ideal control to use when you have long-running AJAX updates.

The next few sections of this chapter look at these new controls and how to use them within your ASP.NET pages.

The ScriptManager Control

Probably the most important control in your ASP.NET AJAX arsenal is the ScriptManager server control, which works with the page to allow for partial page rendering. You use a single ScriptManager control on each page that you want to use the AJAX capabilities provided by ASP.NET 3.5. When placed in conjunction with the UpdatePanel server control, AJAX-enabling your ASP.NET applications can be as simple as adding two server controls to the page and then you are ready to go!

The ScriptManager control takes care of managing the JavaScript libraries that are utilized on your page as well as marshalling the messages back and forth between the server and the client for the partial page rendering process. The marshalling of the messages can be done using either SOAP or JSON through the ScriptManager control.

If you place only a single ScriptManager control on your ASP.NET page, it takes care of loading the JavaScript libraries needed by ASP.NET AJAX. The page for this is presented in Listing 19-8.

Listing 19-8: An ASP.NET page that includes only the ScriptManager control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>The ScriptManager Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
    </div>
    </form>
</body>
</html>
```

From Listing 19-8, you can see that this control is like all other ASP.NET controls and needs only an `ID` and a `runat` attribute to do its work. The page output from this bit of ASP.NET code is presented in Listing 19-9.

Listing 19-9: The page output from the ScriptManager control

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
 The ScriptManager Control
</title></head>
<body>
    <form name="form1" method="post" action="Default2.aspx" id="form1">
<div>
<input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value="" />
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
 value="/wEPDwULLTEzNjQ0OTQ1MDdkZO9dCw2QaeC4D8AwACTbOkD1OX4h" />
</div>

<script type="text/javascript">
//<![CDATA[
var theForm = document.forms['form1'];
if (!theForm) {
    theForm = document.form1;
}
function __doPostBack(eventTarget, eventArgument) {
    if (!theForm.onsubmit || (theForm.onsubmit() != false)) {
        theForm.__EVENTTARGET.value = eventTarget;
        theForm.__EVENTARGUMENT.value = eventArgument;
        theForm.submit();
    }
}
//]]>
</script>
```

Continued

```
<script src="/AJAXWebSite/WebResource.axd?d=o84znEj-
 n4cYi0Wg0pFXCg2&amp;t=633285028458684000" type="text/javascript"></script>
<script src="/AJAXWebSite/ScriptResource.axd?d=
 FETsh5584DXpx8XqIhEM50YSKyR2GkoMoAqraYEDU5_gi1SUmL2Gt7rQTRBAw56lSojJR
 Qe0OjVI8SiYDjmpYmFP0CO8wBFGhtKKJwm2MeE1&amp;t=633285035850304000"
 type="text/javascript"></script>
<script type="text/javascript">
//<![CDATA[
if (typeof(Sys) === 'undefined') throw new Error('ASP.NET AJAX client-side
 framework failed to load.');
//]]>
</script>

<script src="/AJAXWebSite/ScriptResource.axd?d=FETsh5584DXpx8XqIhEM50YSKyR2GkoM
oAqraYEDU5
_gi1SUmL2Gt7rQTRBAw56l7AYfmRViCoO2lZ3XwZ33TGiCt92e_UOqfrP30mdEYnJYs09ulU1xBLj
8TjXOLR1k0&amp;t=633285035850304000" type="text/javascript"></script>
    <div>
        <script type="text/javascript">
//<![CDATA[
Sys.WebForms.PageRequestManager._initialize('ScriptManager1',
 document.getElementById('form1'));
Sys.WebForms.PageRequestManager.getInstance()._updateControls([], [], [], 90);
//]]>
</script>
    </div>

<script type="text/javascript">
//<![CDATA[
Sys.Application.initialize();
//]]>
</script>

</form>
</body>
</html>
```

The page output shows that a number of JavaScript libraries are loaded with the page. You will also notice that the scripts source are dynamically registered and available through the HTTP handler provided through the `ScriptResource.axd` handler.

> *If you are interested in seeing the contents of the JavaScript libraries, you can use the* `src` *attribute's URL in the address bar of your browser and you will be prompted to download the JavaScript file that is referenced. You will be prompted to save the* `ScriptResource.axd` *file, but you can rename it to make use of a* `.txt` *or* `.js` *extension if you wish.*

An interesting point about the ScriptManager is that it deals with the scripts that are sent to the client by taking the extra step to compress them.

The ScriptManagerProxy Control

The ScriptManagerProxy control was actually introduced earlier in this book in Chapter 5 as this control deals specifically with master pages. As with the ScriptManager control in the previous section, you need a single ScriptManager control on each page that is going to be working with ASP.NET AJAX. However, with that said, the big question is what do you do when you are utilizing master pages? Do you need to put the ScriptManager control on the master page and how does this work with the content pages that use the master page?

When you create a new master page from the Add New Item dialog, in addition to an option for a Master Page, there is also an option to add an AJAX Master Page. This option creates the page shown in Listing 19-10.

Listing 19-10: The AJAX Master Page

```
<%@ Master Language="VB" %>

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server" />
        <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">

        </asp:ContentPlaceHolder>
    </div>
    </form>
</body>
</html>
```

This code shows that there is indeed a ScriptManager control on the page and that this page will be added to each and every content page that uses this master page. You do not have to do anything special to a content page to use the ASP.NET AJAX capabilities provided by the master page. Instead, you can create a content page that is no different from any other content page that you might be used to creating.

However, if you are going to want to modify the ScriptManager control that is on the master page in any way, then you have to add a ScriptManagerProxy control to the content page, as shown in Listing 19-11.

Listing 19-11: Adding to the ScriptManager control from the content page

```
<%@ Page Language="VB" MasterPageFile="~/AJAXMaster.master" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1"
 Runat="Server">
 <asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">
     <Scripts>
         <asp:ScriptReference Path="myOtherScript.js" />
     </Scripts>
 </asp:ScriptManagerProxy>
</asp:Content>
```

In this case, the content page adds to the ScriptManager control that is on the master page by interjecting a script reference from the content page. If you use a ScriptManagerProxy control on a content page and there does not happen to be a ScriptManager control on the master page, you will get an error.

The Timer Control

One common task when working with asynchronous postbacks from your ASP.NET pages is that you might want these asynchronous postbacks to occur at specific intervals in time. To accomplish this, you use the Timer control available to you from the AJAX Extensions part of the toolbox. A simple example to demonstrate how this control works involves putting some timestamps on your page and setting postbacks to occur at specific timed intervals. This example is illustrated in Listing 19-12.

Listing 19-12: Using the Timer control

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
       If Not Page.IsPostBack
          Label1.Text = DateTime.Now.ToString()
       End If
    End Sub

    Protected Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs)
       Label1.Text = DateTime.Now.ToString()
    End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Timer Example</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server" />
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
                <asp:Timer ID="Timer1" runat="server" OnTick="Timer1_Tick"
                 Interval="10000">
                </asp:Timer>
            </ContentTemplate>
        </asp:UpdatePanel>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
```

```
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!Page.IsPostBack) {
               Label1.Text = DateTime.Now.ToString();
            }
        }

        protected void Timer1_Tick(object sender, EventArgs e)
        {
            Label1.Text = DateTime.Now.ToString();
        }
    </script>
```

In this case, there are only three controls on the page. The first is the ScriptManager control followed by a Label and the Timer control. When this page loads for the first time, the Label control is populated with the `DateTime` value through the invocation of the `Page_Load` event handler. After this initial load of the `DateTime` value to the Label control, the Timer control takes care of changing this value.

The `OnTick` attribute from the Ticker control enables you to accomplish this task. It points to the function that is triggered when the time span specified in the `Interval` attribute is reached.

The `Interval` attribute is set to `10000`, which is 10,000 milliseconds (remember that there are 1,000 milliseconds to every second). This means, that every 10 seconds an asynchronous postback is performed and the `Timer1_Tick()` function is called.

When you run this page, you will see the time change on the page every 10 seconds.

The UpdatePanel Control

The UpdatePanel server control is an AJAX-specific control that is new in ASP.NET 3.5. The UpdatePanel control is the control that you are likely to use the most when dealing with AJAX. This control preserves the postback model and allows you to perform a partial page render.

The UpdatePanel control is a container control, which means that it does not actually have UI-specific items associated with it. It is a way to trigger a partial page postback and update only the portion of the page that the UpdatePanel specifies.

The <ContentTemplate> Element

There are a couple of ways to deal with the controls on the page that initiate the asynchronous page postbacks. The first is by far the simplest and is shown in Listing 19-13.

Listing 19-13: Putting the triggers inside of the UpdatePanel control

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object,
       ByVal e As System.EventArgs)

       Label1.Text = "This button was clicked on " & DateTime.Now.ToString()
```

Continued

```
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>UpdatePanel Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <asp:Label ID="Label1" runat="server"></asp:Label>
                <br />
                <br />
                <asp:Button ID="Button1" runat="server"
                 Text="Click to initiate async request"
                 OnClick="Button1_Click" />
            </ContentTemplate>
        </asp:UpdatePanel>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "This button was clicked on " + DateTime.Now.ToString();
    }
</script>
```

In this case, the Label and Button server controls are contained within the UpdatePanel server control. The `<asp:UpdatePanel>` element has two possible sub-elements: `<ContentTemplate>` and the `<Triggers>` elements. Any content that needs to be changed with the asynchronous page postbacks should be contained within the `<ContentTemplate>` section of the UpdatePanel control.

By default, any type of control trigger (something that would normally trigger a page postback) that is contained within the `<ContentTemplate>` section instead causes the asynchronous page postback. That means, in the case of Listing 19-13, the button on the page will trigger an asynchronous page postback instead of a full-page postback. Each click on the button changes the time displayed in the Label control.

The <Triggers> Element

Listing 19-13 demonstrates one of the big issues with this model: When the asynchronous postback occurs, you are not only sending the date/time value for the Label control, but you are also sending back the entire code for the button that is on the page.

```
265|updatePanel|UpdatePanel1|
    <span id="Label1">This button was clicked on 11/18/2007 11:45:21 AM</span>
    <br />
    <br />
    <input type="submit" name="Button1" value="Click to initiate async request"
     id="Button1" />
|164|hiddenField|__VIEWSTATE|/wEPDwUKLTU2NzQ4MzIwMw9kFgICBA9kFgICAw9kFgJmD2QWAgIBDw
8WAh4EVGV4dAUxVGhpcyBidXR0b24gd2FzIGNsaWNrZWQgb24gMTEvMTgvMjAwNyAxMTo0NToyMSBBTWRkZ
KJIG4WwhyQvUwPCX4PxI5FEUFtC|48|hiddenField|__EVENTVALIDATION|/wEWAgL43YXdBwKM54rGBl
I52OYVl/McOV61BYd/3wSj+RkD|0|asyncPostBackControlIDs|||0|postBackControlIDs|||13|
updatePanelIDs||tUpdatePanel1|0|childUpdatePanelIDs|||12|panelsToRefreshIDs||
UpdatePanel1|2|asyncPostBackTimeout||90|22|formAction||SimpleUpdatePanel.aspx|11|
pageTitle||UpdatePanel|
```

This bit of code that is sent back to the client via the asynchronous postback shows that the entire section contained within the UpdatePanel control is reissued. You can slim down your pages by including only the portions of the page that are actually updating. If you take the button outside of the `<ContentTemplate>` section of the UpdatePanel control, then you have to include a `<Triggers>` section within the control.

The reason for this is that while the content that you want to change with the asynchronous postback is all contained within the `<ContentTemplate>` section, you have to tie up a page event to cause the postback to occur. This is how the `<Triggers>` section of the UpdatePanel control is used. You use this section of the control to specify the various triggers that initiate an asynchronous page postback. Using the `<Triggers>` element within the UpdatePanel control, you can rewrite Listing 19-13 as shown in Listing 19-14.

Listing 19-14: Using a trigger to cause the asynchronous page postback

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object,
       ByVal e As System.EventArgs)

       Label1.Text = "This button was clicked on " & DateTime.Now.ToString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>UpdatePanel</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <asp:Label ID="Label1" runat="server"></asp:Label>
            </ContentTemplate>
```

Continued

```
            <Triggers>
                <asp:AsyncPostBackTrigger ControlID="Button1" EventName="Click" />
            </Triggers>
        </asp:UpdatePanel>
        <br />
        <br />
        <asp:Button ID="Button1" runat="server"
         Text="Click to initiate async request"
         OnClick="Button1_Click" />
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">

    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "This button was clicked on " + DateTime.Now.ToString();
    }
</script>
```

In this case, the Button control and the HTML elements are outside of the `<ContentTemplate>` section of the UpdatePanel control and therefore will not be sent back to the client for each asynchronous page postback. The only item contained in the `<ContentTemplate>` section is the only item on the page that needs to change with the postbacks — the Label control. Tying this all together is the `<Triggers>` section.

The `<Triggers>` section can contain two possible controls: AsyncPostBackTrigger and PostBackTrigger. In this case, the AsyncPostBackTrigger is used. The PostBackTrigger control will cause a full page postback, whereas the AsyncPostBackTrigger control will cause only an asynchronous page postback (obviously as described by the names of the controls).

As you can see from the example in Listing 19-14, which uses the AsyncPostBackTrigger element, only two attributes are used to tie the Button control to the trigger for the asynchronous postback: the `ControlID` and the `EventName` attributes. The control you want to act as the initiator of the asynchronous page postback is put here (the control's name as specified by the control's `ID` attribute). The `EventName` attribute's value is the name of the event for the control that is specified in the `ControlID` that you want to be called in the asynchronous request from the client. In this case, the Button control's `Click()` event is called and this is the event that changes the value of the control that resides within the `<ContentTemplate>` section of the UpdatePanel control.

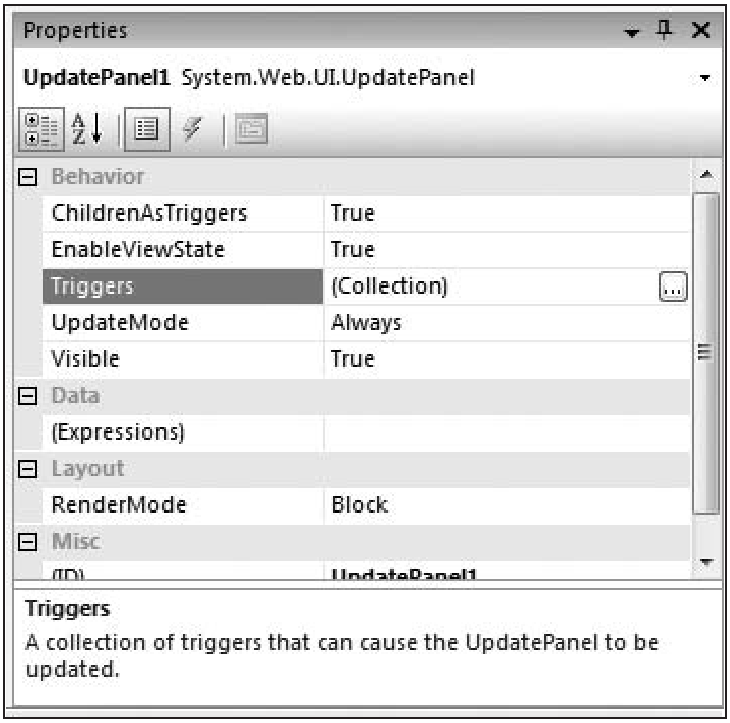
Running this page and clicking on the button gives you a smaller asynchronous response back to the client.

```
108|updatePanel|UpdatePanel1|
   <span id="Label1">This button was clicked on 11/18/2007 11:58:56 AM</span>
|164|hiddenField|__VIEWSTATE|/wEPDwUKMjA2NjQ2MDYzNw9kFgICBA9kFgICAw9kFgJmD2QWAgIBDw
8WAh4EVGV4dAUxVGhpcyBidXR0b24gd2FzIGNsaWNrZWQgb24gMTEvMTgvMjAwNyAxMTo1ODo1NiBBTWRkZ
PJA9uj9wwRaasgTrZo85rVvLnoi|48|hiddenField|__EVENTVALIDATION|/wEWAgKK3YDTDAKM54rGBq
rbjV4/u4ks3aKsn7Xz8xNFE8G/|7|asyncPostBackControlIDs||Button1|0|postBackControlIDs|
||13|updatePanelIDs||tUpdatePanel1|0|childUpdatePanelIDs|||12|panelsToRefreshIDs||
UpdatePanel1|2|asyncPostBackTimeout||90|22|formAction||SimpleUpdatePanel.aspx|11|
pageTitle||UpdatePanel|
```

Although not considerably smaller than the previous example, it is smaller and the size similarity is really due to the size of the page used in this example (pages that are more voluminous would show more dramatic improvements). Pages with heavy content associated with them can show some dramatic size reductions depending on how you structure your pages with the UpdatePanel control.

Building Triggers Using Visual Studio 2008

If you like to work on the design surface of Visual Studio when building your ASP.NET pages, you will find that there is good support for building your ASP.NET AJAX pages, including the creation of triggers in the UpdatePanel control. To see this in action, place a single UpdatePanel server control on your page and view the control in the Properties dialog within Visual Studio. The Triggers item in the list has a button next to it that allows you to modify the items associated with it. This is illustrated in Figure 19-11.



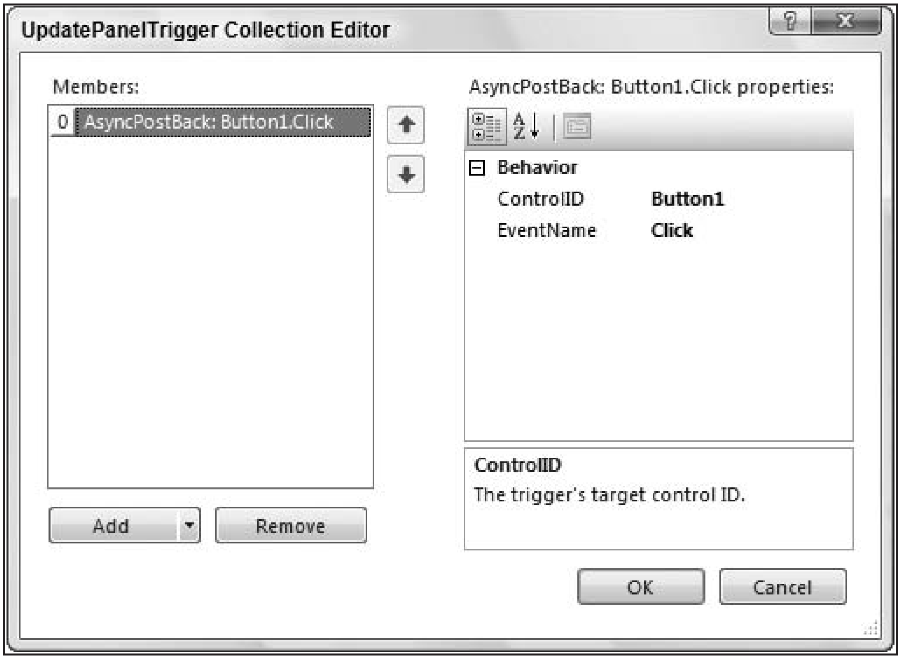
ChildrenAsTriggers True
EnableViewState True
Triggers (Collection)
UpdateMode Always
Visible True
(Expressions)
RenderMode Block
(ID) UpdatePanel1

Triggers
A collection of triggers that can cause the UpdatePanel to be updated.

Figure 19-11

Clicking on the button in the Properties dialog launches the UpdatePanelTrigger Collection Editor, as shown in Figure 19-12. This editor allows you to add any number of triggers and to associate them to a control and a control event very easily.

Clicking the OK button here adds the trigger to the `<Triggers>` section of your UpdatePanel control.


UpdatePanelTrigger Collection Editor
Members:
0 AsyncPostBack: Button1.Click
AsyncPostBack: Button1.Click properties:
Behavior
Add
Remove
OK
Cancel

ControlID Button1

EventName Click

ControlID

The trigger's target control ID.

Figure 19-12

The UpdateProgress Control

The final server control in the AJAX Extensions section of Visual Studio 2008 is the UpdateProgress control. Some asynchronous postbacks take some time to execute because of the size of the response or because of the computing time required to get a result together to send back to the client. The UpdateProgress control allows you to provide a visual signifier to the clients to show that indeed work is being done and they will get results soon (and that the browser simply didn't just lock up).

Listing 19-15 shows a textual implementation of the UpdateProgress control.

Listing 19-15: Using the UpdateProgess control to show a text message to the client

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object,
       ByVal e As System.EventArgs)

       System.Threading.Thread.Sleep(10000)
       Label1.Text = "This button was clicked on " & DateTime.Now.ToString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

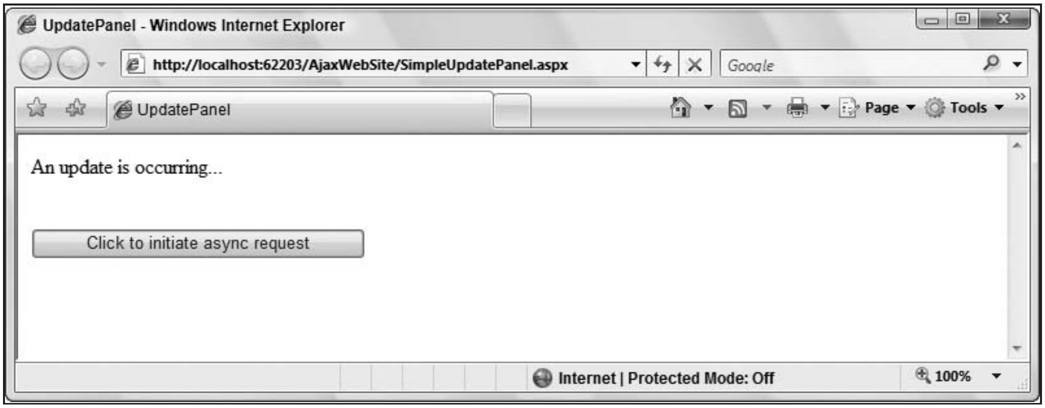
```
    <title>UpdatePanel</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <asp:UpdateProgress ID="UpdateProgress1" runat="server">
            <ProgressTemplate>
                An update is occurring...
            </ProgressTemplate>
        </asp:UpdateProgress>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
            <ContentTemplate>
                <asp:Label ID="Label1" runat="server"></asp:Label>
            </ContentTemplate>
            <Triggers>
                <asp:AsyncPostBackTrigger ControlID="Button1" EventName="Click" />
            </Triggers>
        </asp:UpdatePanel>
        <br />
        <br />
        <asp:Button ID="Button1" runat="server"
         Text="Click to initiate async request"
         OnClick="Button1_Click" />
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>
<script runat="server">

    protected void Button1_Click(object sender, EventArgs e)
    {
        System.Threading.Thread.Sleep(10000);
        Label1.Text = "This button was clicked on " + DateTime.Now.ToString();
    }
</script>
```

To add some delay to the response (in order to simulate a long running computer process) the Thread.Sleep() method is called. From here, you add an UpdateProgess control to the part of the page where you want the update message to be presented. In this case, the UpdateProgress control was added above the UpdatePanel server control. This control does not go inside the UpdatePanel control; instead, it sits outside of the control. However, like the UpdatePanel control, the UpdateProgress control is a template control.

The UpdateProgress control has only a single sub-element: the `<ProgressTemplate>` element. Whatever you place in this section of the control will appear when the UpdateProgress control is triggered. In this case, the only item present in this section of the control is some text. When you run this page, you will get the update shown in Figure 19-13.



An update is occurring...

UpdatePanel - Windows Internet Explorer
http://localhost:62203/AjaxWebSite/SimpleUpdatePanel.aspx
Google
UpdatePanel
Page
Tools
Click to initiate async request
Internet | Protected Mode: Off
100%

Figure 19-13

The text will appear immediately in this case and will not disappear until the asynchronous postback has finished. The code you put in the `<ProgressTemplate>` section is actually contained in the page, but its display is turned off through CSS.

```
<div id="UpdateProgress1" style="display:none;">
   An update is occurring...
</div>
```

Controlling When the Message Appears

Right now, the UpdateProgress appears as soon as the button is clicked. However, some of your processes might not take that long and you might not always want a progress notification going out to the client. The UpdateProgress control includes a `DisplayAfter` attribute, which allows you to control when the progress update message appears. The use of the `DisplayAfter` attribute is shown in Listing 19-16.

Listing 19-16: Using the DisplayAfter attribute

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server" DisplayAfter="5000">
   <ProgressTemplate>
      An update is occurring...
   </ProgressTemplate>
</asp:UpdateProgress>
```

The value of the `DisplayAfter` property is a number that represents the number of milliseconds that the UpdateProgress control will wait until it displays what is contained within the `<ProgressTemplate>` section. The code in Listing 19-16 specifies that the text found in the `<ProgressTemplate>` section will not be displayed for 5,000 milliseconds (5 seconds).

Adding an Image to the <ProcessTemplate>

The previous examples which make use of the UpdateProgress control use this control with text, but you can put anything you want within this template control. For instance, you can put a spinning wheel image that will show the end user that the request is being processed. The use of the image is shown in Listing 19-17.

Listing 19-17: Using an image in the <ProcessTemplate> section

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server" DisplayAfter="5000">
   <ProgressTemplate>
      <asp:Image ID="Image1" runat="server" ImageUrl="~/spinningwheel.gif" />
   </ProgressTemplate>
</asp:UpdateProgress>
```

Just as the in the text approach, the code for the image is already placed on the client's page instance and is just turned off via CSS.

```
<div id="UpdateProgress1" style="display:none;">
   <img id="Image1" src="spinningwheel.gif" style="border-width:0px;" />
</div>
```

Using Multiple UpdatePanel Controls

So far, this chapter has showed you how to work with a single UpdatePanel control, but it is important to realize that you can have multiple UpdatePanel controls on a single page. This, in the end, will give you the ability to control the output to specific regions of the page when you want.

An example of using more than a single UpdatePanel control is presented in Listing 19-18.

Listing 19-18: Using more than one UpdatePanel control

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object,
       ByVal e As System.EventArgs)

       Label1.Text = "Label1 was populated on " & DateTime.Now.ToString()
       Label2.Text = "Label2 was populated on " & DateTime.Now.ToString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Multiple UpdatePanel Controls</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <asp:Label ID="Label1" runat="server"></asp:Label>
            </ContentTemplate>
            <Triggers>
                <asp:AsyncPostBackTrigger ControlID="Button1" EventName="Click" />
            </Triggers>
        </asp:UpdatePanel>
```

Continued

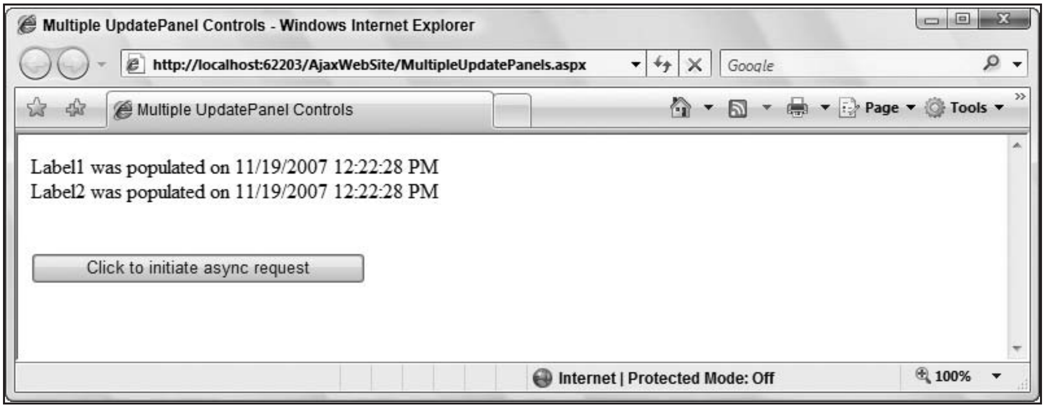
```
            <asp:UpdatePanel ID="UpdatePanel2" runat="server">
                <ContentTemplate>
                    <asp:Label ID="Label2" runat="server"></asp:Label>
                </ContentTemplate>
            </asp:UpdatePanel>
            <br />
            <br />
            <asp:Button ID="Button1" runat="server"
             Text="Click to initiate async request"
             OnClick="Button1_Click" />
        </div>
        </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "Label1 was populated on " + DateTime.Now;
        Label2.Text = "Label2 was populated on " + DateTime.Now;
    }
</script>
```

This is an interesting page. There are two UpdatePanel controls on the page: `UpdatePanel1` and `UpdatePanel2`. Each of these controls contains a single Label control that at one point can take a date/time value from a server response.

The `UpdatePanel1` control has an associated trigger: the Button control on the page. When this button is clicked, the `Button1_Click()` event triggers and does its job. If you run this page, both of the UpdatePanel controls are updated according to the `Button1_Click()` event. This is illustrated in Figure 19-14.


Multiple UpdatePanel Controls - Windows Internet Explorer
http://localhost:62203/AjaxWebSite/MultipleUpdatePanels.aspx
Google
Multiple UpdatePanel Controls
Page
Tools
Click to initiate async request
Internet | Protected Mode: Off
100%

Label1 was populated on 11/19/2007 12:22:28 PM
Label2 was populated on 11/19/2007 12:22:28 PM

Figure 19-14

Both UpdatePanel sections were updated with the `button-click` event because, by default, all UpdatePanel controls on a single page update with *each* asynchronous postback that occurs. This means that the postback that occurred with the `Button1` button control also causes a postback to occur with the `UpdatePanel2` control.

You can actually control this behavior through the UpdatePanel's `UpdateMode` property. The `UpdateMode` property can take two possible enumerations — `Always` and `Conditional`. If you do not set this property, it uses the value of `Always`, meaning that each UpdatePanel control always updates with each asynchronous request.

The other option is to set the property to `Conditional`. This means that the UpdatePanel updates only if one of the trigger conditions is met. For an example of this, change the UpdatePanel controls on the page so that they are now using an `UpdateMode` of `Conditional`, as shown in Listing 19-19.

Listing 19-19: Using more than one UpdatePanel control

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object,
       ByVal e As System.EventArgs)

       Label1.Text = "Label1 was populated on " & DateTime.Now.ToString()
       Label2.Text = "Label2 was populated on " & DateTime.Now.ToString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Multiple UpdatePanel Controls</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
            <ContentTemplate>
                <asp:Label ID="Label1" runat="server"></asp:Label>
            </ContentTemplate>
            <Triggers>
                <asp:AsyncPostBackTrigger ControlID="Button1" EventName="Click" />
            </Triggers>
        </asp:UpdatePanel>
        <asp:UpdatePanel ID="UpdatePanel2" runat="server" UpdateMode="Conditional">
            <ContentTemplate>
                <asp:Label ID="Label2" runat="server"></asp:Label>
            </ContentTemplate>
        </asp:UpdatePanel>
        <br />
        <br />
```

Continued

```
            <asp:Button ID="Button1" runat="server"
             Text="Click to initiate async request"
             OnClick="Button1_Click" />
        </div>
        </form>
    </body>
    </html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "Label1 was populated on " + DateTime.Now;
        Label2.Text = "Label2 was populated on " + DateTime.Now;
    }
</script>
```

Now that both of the UpdatePanel controls are set to have an `UpdateMode` of `Conditional`, when running this page, you will see the results presented in Figure 19-15.

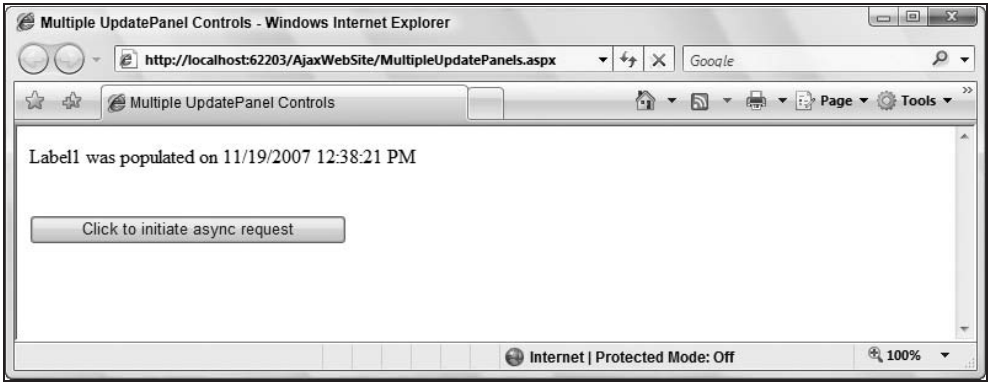

Multiple UpdatePanel Controls - Windows Internet Explorer
http://localhost:62203/AjaxWebSite/MultipleUpdatePanels.aspx
Google
Multiple UpdatePanel Controls
Page
Tools
Label1 was populated on 11/19/2007 12:38:21 PM
Click to initiate async request
Internet | Protected Mode: Off
100%

Figure 19-15

In this case, only the right Label control, `Label1`, was updated with the asynchronous request even though the `Button1_Click()` event tries to change the values of both `Label1` and `Label2`. The reason for this is that the UpdatePanel2 control had no trigger that was met.

Summary

ASP.NET AJAX, although it is in its infancy, is an outstanding technology and will fundamentally change the way Web application development is approached. No longer do you need to completely tear down a page and rebuild it for each and every request. Instead, you are able to rebuild the pages slowly in sections as the end user requests them. The line between the thin-client world and the thick-client world just got a lot more opaque.

This chapter took a look at the core foundation of ASP.NET AJAX that is available with the default install of Visual Studio 2008. Beyond that, there is much more. A big addition is the ASP.NET AJAX Control Toolkit, the focus of the next chapter.

20

ASP.NET AJAX Control Toolkit

ASP.NET AJAX applications were introduced in the previous chapter. With the install of the .NET Framework 3.5 and through using Visual Studio 2008, you will find a few controls available that allow you to build ASP.NET applications with AJAX capabilities. This is the framework to take your applications to the next level because there is so much that can be accomplished with it, including adding specific AJAX capabilities to your user and custom server controls. Every AJAX enhancement added to your application will make your application seem more fluid and responsive to the end user.

You might be wondering where the big new AJAX-enabled server controls are for this edition of Visual Studio 2008 if this is indeed a new world for building controls. The reason you do not see a new section of AJAX server controls is that Microsoft's has treated them as an open-source project instead of just blending them into Visual Studio 2008.

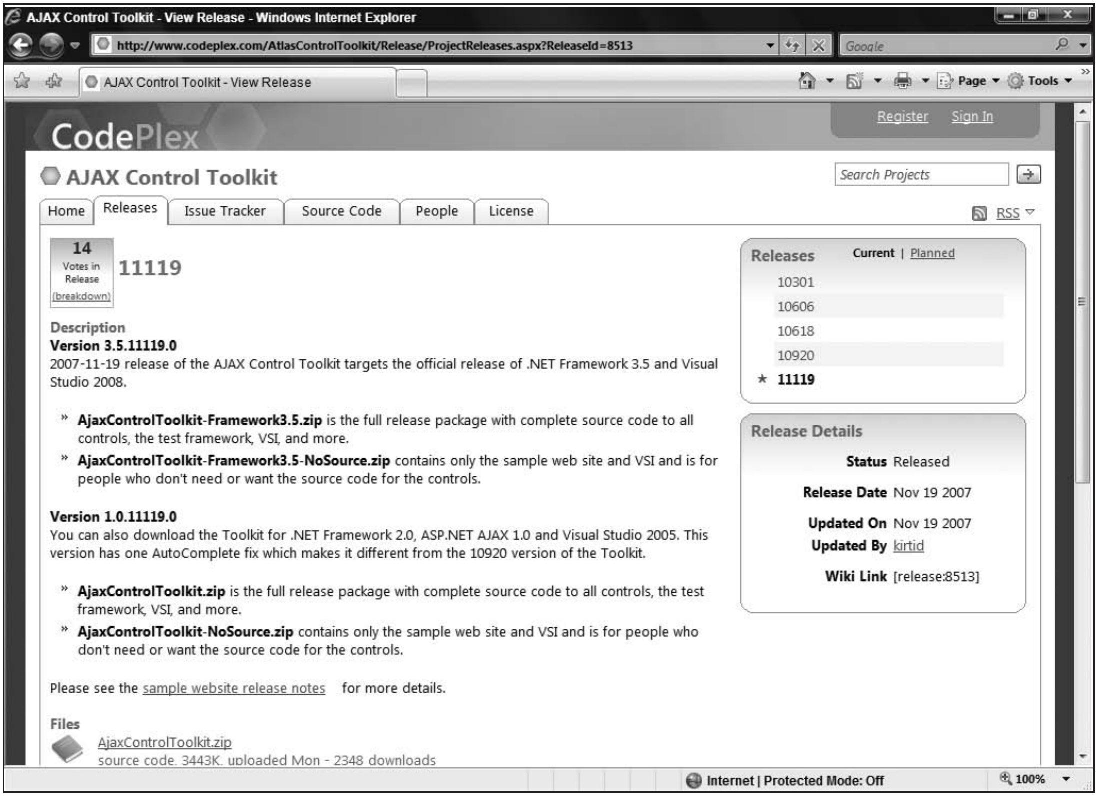
Developers at Microsoft and in the community have been working on a series of AJAX-capable server controls that you can use in your ASP.NET applications. These controls are collectively called the ASP.NET AJAX Control Toolkit. You will find a link to the control toolkit for download at `www.asp.net/AJAX`. When you choose to download the control toolkit from this page, you will be directed to the ASP.NET AJAX Control Toolkit's page on CodePlex at `www.codeplex.com/Release/ProjectReleases.aspx?ProjectName=AtlasControlToolkit`. This page is shown in Figure 20-1.

Downloading and Installing

Since the ASP.NET AJAX Control Toolkit is not part of the default install of Visual Studio 2008, you have to set up the controls yourself. Again, the control toolkit's site on CodePlex offers a couple of options to you.

First off, you are able to download a control toolkit that is specifically targeted at Visual Studio 2008 or Visual Studio 2005. This chapter focuses on using the control toolkit with Visual Studio 2008.

The CodePlex page for this project offers two ways to get what you are after. The ASP.NET AJAX Control Toolkit can be downloaded as source code or as a compiled DLL.


AJAX Control Toolkit - View Release - Windows Internet Explorer
http://www.codeplex.com/AtlasControlToolkit/Release/ProjectReleases.aspx?ReleaseId=8513
Google
AJAX Control Toolkit - View Release
Page
Tools
CodePlex
Register
Sign In
AJAX Control Toolkit
Search Projects
Home
Releases
Issue Tracker
Source Code
People
License
RSS
14
Votes in Release
(breakdown)
11119
Releases
Current | Planned
10301
10606
10618
10920
★ 11119
Internet | Protected Mode: Off
100%

Description

Version 3.5.11119.0

2007-11-19 release of the AJAX Control Toolkit targets the official release of .NET Framework 3.5 and Visual Studio 2008.

- **AjaxControlToolkit-Framework3.5.zip** is the full release package with complete source code to all controls, the test framework, VSI, and more.
- **AjaxControlToolkit-Framework3.5-NoSource.zip** contains only the sample web site and VSI and is for people who don't need or want the source code for the controls.

Version 1.0.11119.0

You can also download the Toolkit for .NET Framework 2.0, ASP.NET AJAX 1.0 and Visual Studio 2005. This version has one AutoComplete fix which makes it different from the 10920 version of the Toolkit.

- **AjaxControlToolkit.zip** is the full release package with complete source code to all controls, the test framework, VSI, and more.
- **AjaxControlToolkit-NoSource.zip** contains only the sample web site and VSI and is for people who don't need or want the source code for the controls.

Please see the sample website release notes for more details.

Files

AjaxControlToolkit.zip
source code, 3443K, uploaded Mon - 2348 downloads

Release Details

Status Released
Release Date Nov 19 2007
Updated On Nov 19 2007
Updated By kirtid
Wiki Link [release:8513]

Figure 20-1

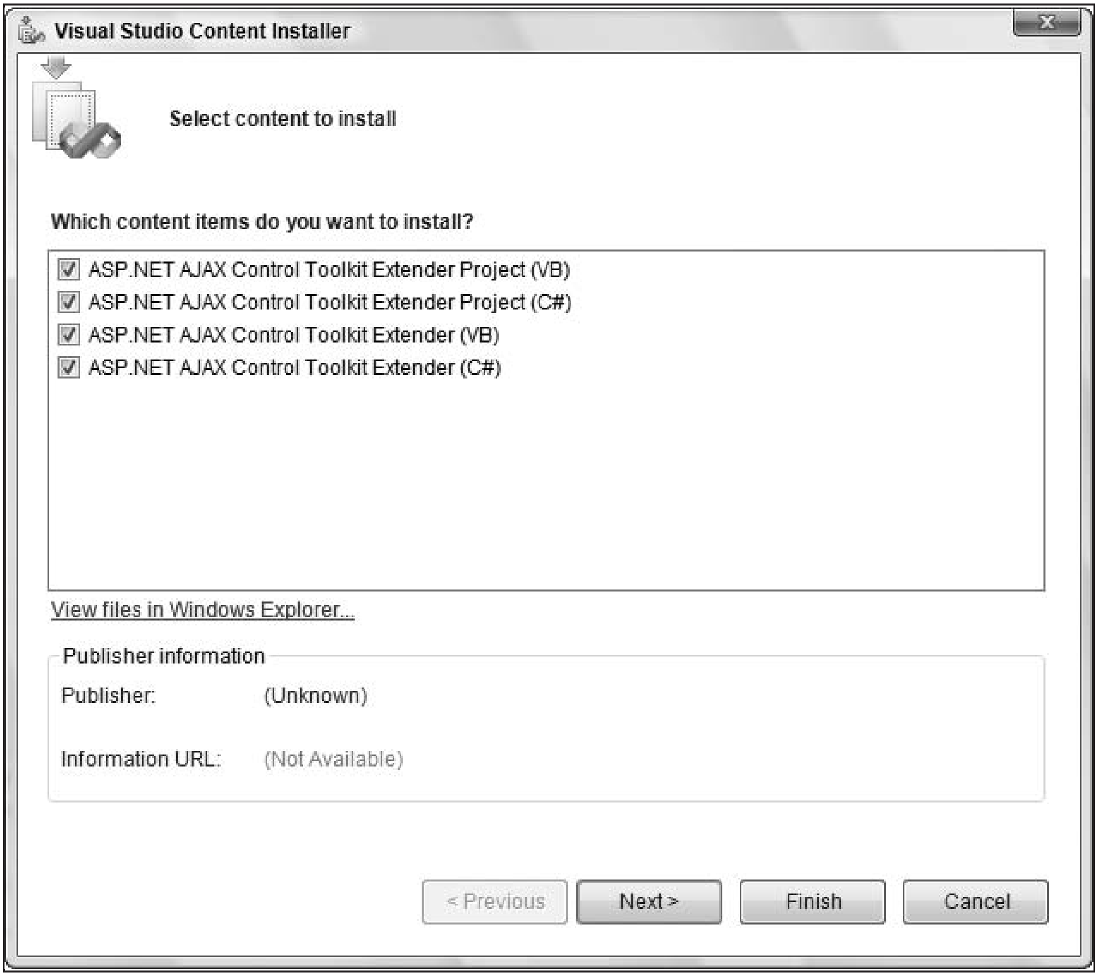
The source control option allows you to take the code for the controls and ASP.NET AJAX extenders and change the behavior of the controls or extenders yourself. The DLL option is a single Visual Studio installer file and a sample application.

There are a couple of parts to the install. One part provides a series of new controls that were built with AJAX capabilities in mind. Another part is a series of control extenders (extensions upon pre-existing controls). You will also find some new template additions for your Visual Studio 2008 instance.

To get set up, download the `.zip` file from the CodePlex site and unzip it where you want on your machine. The following sections show you how to work with the various parts provided from the Control Toolkit.

New Visual Studio Templates

Included in the download is a `.vsi` file (Visual Studio Installer) called `AJAXControlExtender.vsi`. This installer will install four new Visual Studio projects for your future use. The installer options are shown in Figure 20-2.


Visual Studio Content Installer
Select content to install

Which content items do you want to install?

☑ ASP.NET AJAX Control Toolkit Extender Project (VB)
☑ ASP.NET AJAX Control Toolkit Extender Project (C#)
☑ ASP.NET AJAX Control Toolkit Extender (VB)
☑ ASP.NET AJAX Control Toolkit Extender (C#)

View files in Windows Explorer...
Publisher information

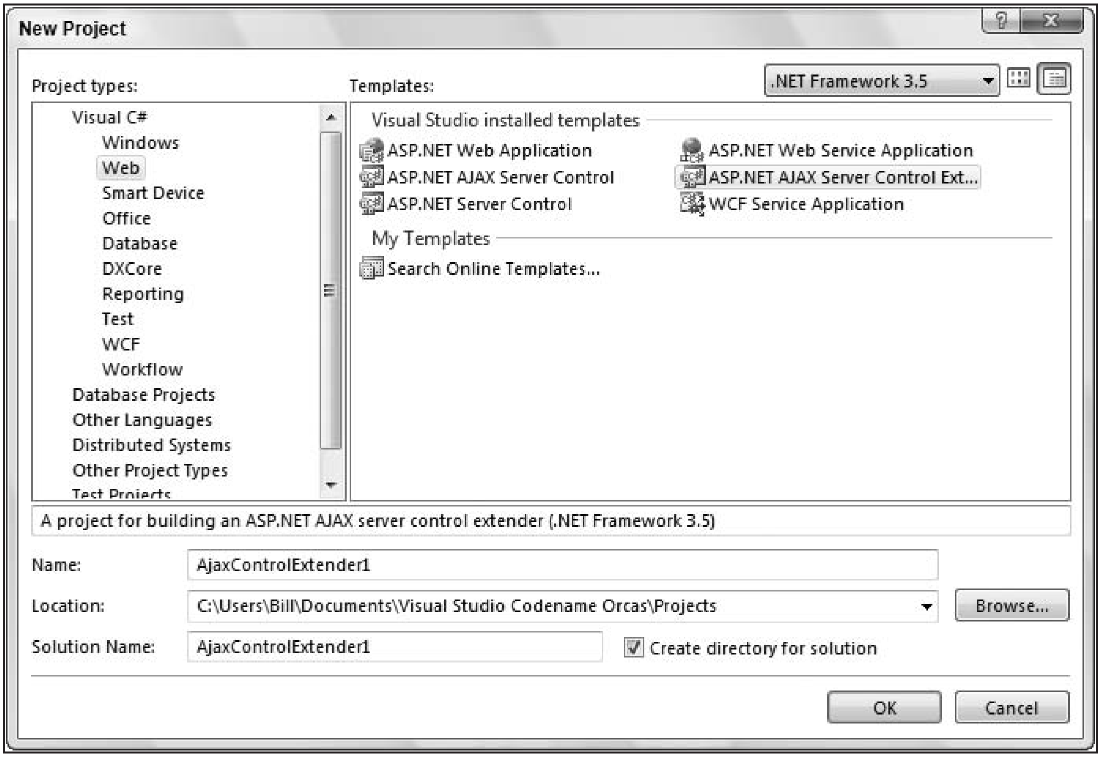
Publisher: (Unknown)

Information URL: (Not Available)

< Previous
Next >
Finish
Cancel

Figure 20-2

After installing these options, you have access to the new project types this provides as illustrated in Figure 20-3.


New Project
Project types:
Visual C#
Windows
Web
Smart Device
Office
Database
DXCore
Reporting
Test
WCF
Workflow
Database Projects
Other Languages
Distributed Systems
Other Project Types
Test Projects
Templates:
Visual Studio installed templates
ASP.NET Web Application
ASP.NET Web Service Application
ASP.NET AJAX Server Control
ASP.NET AJAX Server Control Ext...
ASP.NET Server Control
WCF Service Application
My Templates
Search Online Templates...
Browse...
OK
Cancel

.NET Framework 3.5

A project for building an ASP.NET AJAX server control extender (.NET Framework 3.5)

Name: AjaxControlExtender1

Location: C:\Users\Bill\Documents\Visual Studio Codename Orcas\Projects

Solution Name: AjaxControlExtender1

☑ Create directory for solution

Figure 20-3

Adding the New Controls to the VS2008 Toolbox

In addition to new project types, you can also add the new controls to your Visual Studio 2008 Toolbox. To accomplish this task, right-click in the Toolbox and select Add Tab from the provided menu. Name the tab as you wish, though, for this example the tab was named AJAX Controls.

With the new tab in your Toolbox, right-click the tab and select Choose Items from the provided menu. This action is illustrated in Figure 20-4.

Select Choose Items from the menu to open the Choose Toolbox Items dialog. From here, you want to select `AjaxControlToolkit.dll` from the sample application's Bin folder. Remember that the SampleWebSite is a component that was downloaded and unzipped as part of the Control Toolkit. When you find the DLL and click Open. The Choose Toolbox Items dialog changes to include the controls that are contained within this DLL. The controls are highlighted in the dialog and are already selected for you (as shown in Figure 20-5).

From here, click OK and the ASP.NET AJAX Control Toolkit's new controls will be added to your Visual Studio Toolbox. The end result is presented in Figure 20-6.

You will find that there are more than 35 new controls and extenders added to the Toolbox for you to utilize in your ASP.NET applications.

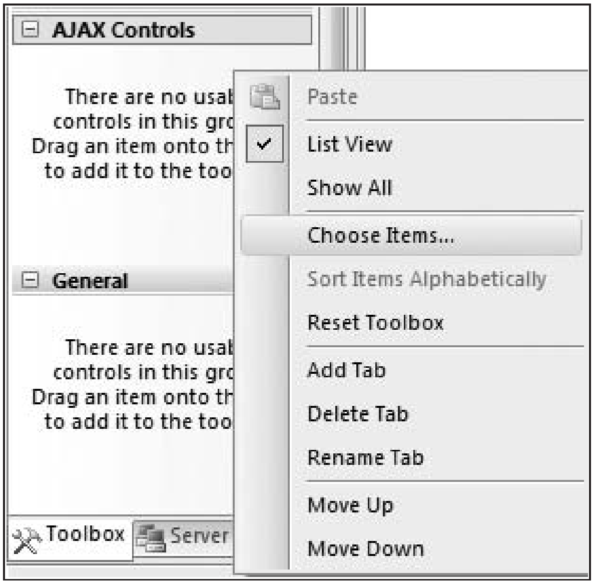
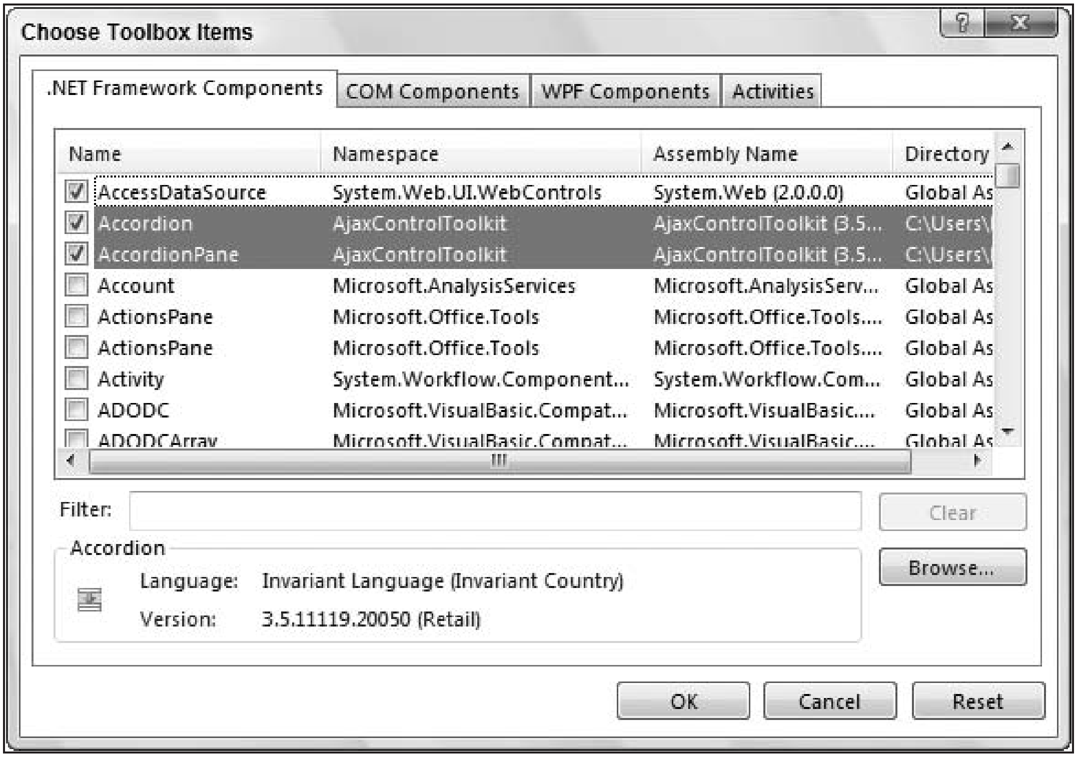

AJAX Controls
There are no usab
controls in this gro
Drag an item onto th
to add it to the too
General
There are no usab
controls in this gro
Drag an item onto th
to add it to the too
Toolbox
Server
Paste
List View
Show All
Choose Items...
Sort Items Alphabetically
Reset Toolbox
Add Tab
Delete Tab
Rename Tab
Move Up
Move Down

Figure 20-4


Choose Toolbox Items
.NET Framework Components
COM Components
WPF Components
Activities
Filter:
Clear
Browse...
OK
Cancel
Reset

	Name	Namespace	Assembly Name	Directory
☑	AccessDataSource	System.Web.UI.WebControls	System.Web (2.0.0.0)	Global As
☑	Accordion	AjaxControlToolkit	AjaxControlToolkit (3.5...	C:\Users\
☑	AccordionPane	AjaxControlToolkit	AjaxControlToolkit (3.5...	C:\Users\
☐	Account	Microsoft.AnalysisServices	Microsoft.AnalysisServ...	Global As
☐	ActionsPane	Microsoft.Office.Tools	Microsoft.Office.Tools....	Global As
☐	ActionsPane	Microsoft.Office.Tools	Microsoft.Office.Tools....	Global As
☐	Activity	System.Workflow.Component...	System.Workflow.Com...	Global As
☐	ADODC	Microsoft.VisualBasic.Compat...	Microsoft.VisualBasic....	Global As
☐	ADODCArray	Microsoft.VisualBasic.Compat...	Microsoft.VisualBasic....	Global As

Accordion

Language: Invariant Language (Invariant Country)

Version: 3.5.11119.20050 (Retail)

Figure 20-5

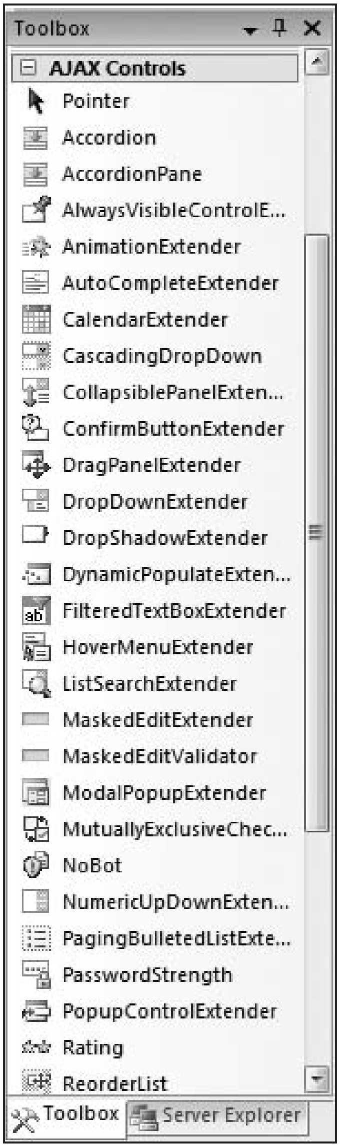

Toolbox
AJAX Controls
Pointer
Accordion
AccordionPane
AlwaysVisibleControlE...
AnimationExtender
AutoCompleteExtender
CalendarExtender
CascadingDropDown
CollapsiblePanelExten...
ConfirmButtonExtender
DragPanelExtender
DropDownExtender
DropShadowExtender
DynamicPopulateExten...
FilteredTextBoxExtender
HoverMenuExtender
ListSearchExtender
MaskedEditExtender
MaskedEditValidator
ModalPopupExtender
MutuallyExclusiveChec...
NoBot
NumericUpDownExten...
PagingBulletedListExte...
PasswordStrength
PopupControlExtender
Rating
ReorderList
Toolbox
Server Explorer

Figure 20-6

The ASP.NET AJAX Controls

The number of controls and extenders available from the Control Toolkit is large. As stated, there are more than 35 controls and extenders at your disposal. This section will look at these new items and how you can use them in your ASP.NET applications.

When you add an ASP.NET AJAX server control to your page, one thing you will notice is that a number of DLLs focused on localization into a number of languages have been added to the Bin folder of your solution. All the resource files have been organized into language folders within the folder. An example of what you will find is presented in Figure 20-7.

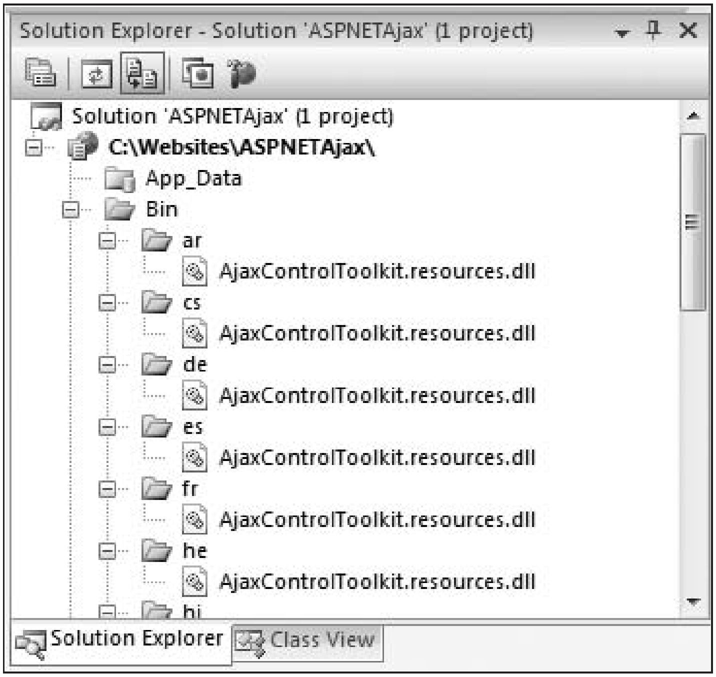

Solution Explorer - Solution 'ASPNETAjax' (1 project)
Solution 'ASPNETAjax' (1 project)
C:\Websites\ASPNETAjax\
App_Data
Bin
ar
AjaxControlToolkit.resources.dll
cs
AjaxControlToolkit.resources.dll
de
AjaxControlToolkit.resources.dll
es
AjaxControlToolkit.resources.dll
fr
AjaxControlToolkit.resources.dll
he
AjaxControlToolkit.resources.dll
hi
Solution Explorer
Class View

Figure 20-7

Looking at one of the DLLs with Lutz Roeder's .NET Reflector tool (`www.aisto.com/roeder/dotnet/`), you will notice that they are focused on the client-side localization required by many applications. As an example, the `AjaxControlToolkit.resources.dll` for the Russian language within Reflector is shown in Figure 20-8.

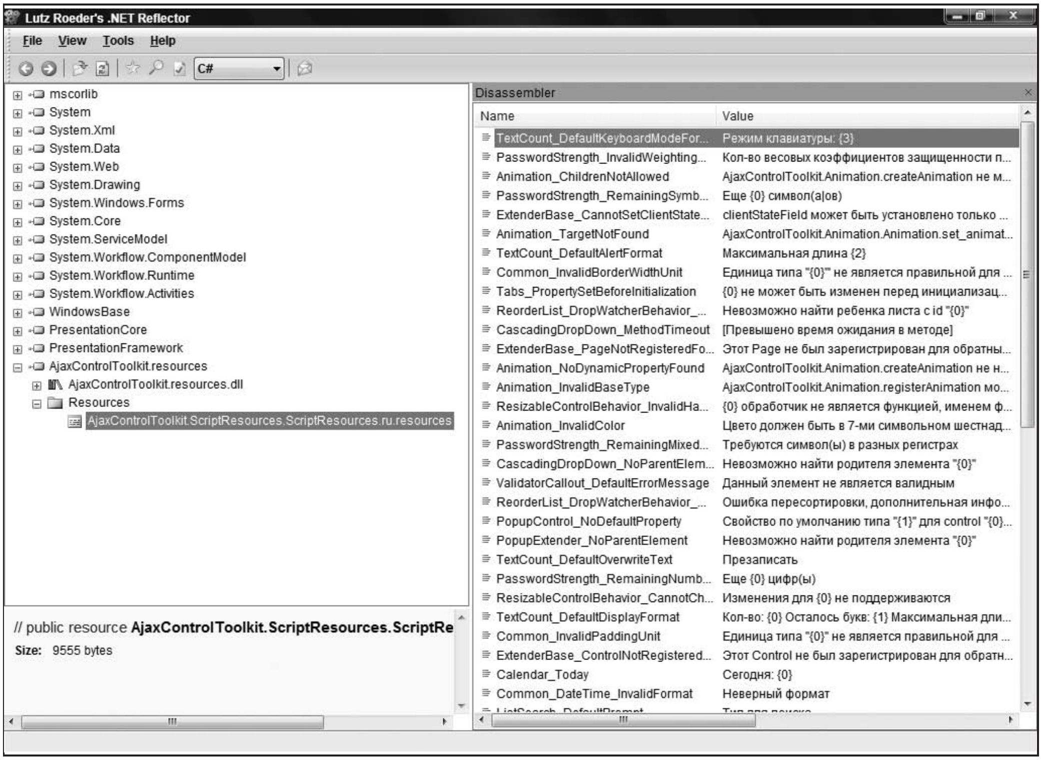
In addition to the localization DLLs added to your project, the ASP.NET AJAX control is added just as any other custom server control in ASP.NET. Listing 20-1 shows what your ASP.NET page looks like after the addition of a single ASP.NET AJAX control to it.

Listing 20-1: Changes to the ASP.NET page after adding an ASP.NET AJAX control

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeFile="Default.aspx.cs" Inherits="_Default" %>

<%@ Register Assembly="AjaxControlToolkit"
    Namespace="AjaxControlToolkit" TagPrefix="cc1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Continued

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
       <asp:ScriptManager ID="ScriptManager1" runat="server" />

        <cc1:AlwaysVisibleControlExtender
         ID="AlwaysVisibleControlExtender1" runat="server"
         TargetControlID="TextBox1">
        </cc1:AlwaysVisibleControlExtender>

        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </div>
    </form>
</body>
</html>
```


Lutz Roeder's .NET Reflector
File
View
Tools
Help
C#
mscorlib
System
System.Xml
System.Data
System.Web
System.Drawing
System.Windows.Forms
System.Core
System.ServiceModel
System.Workflow.ComponentModel
System.Workflow.Runtime
System.Workflow.Activities
WindowsBase
PresentationCore
PresentationFramework
AjaxControlToolkit.resources
AjaxControlToolkit.resources.dll
Resources
AjaxControlToolkit.ScriptResources.ScriptResources.ru.resources
Disassembler

// public resource **AjaxControlToolkit.ScriptResources.ScriptRe**

Size: 9555 bytes

Name	Value
TextCount_DefaultKeyboardModeFor...	Режим клавиатуры: {3}
PasswordStrength_InvalidWeighting...	Кол-во весовых коэффициентов защищенности п...
Animation_ChildrenNotAllowed	AjaxControlToolkit.Animation.createAnimation не м...
PasswordStrength_RemainingSymb...	Еще {0} символ(а|ов)
ExtenderBase_CannotSetClientState...	clientStateField может быть установлено только ...
Animation_TargetNotFound	AjaxControlToolkit.Animation.Animation.set_animat...
TextCount_DefaultAlertFormat	Максимальная длина {2}
Common_InvalidBorderWidthUnit	Единица типа "{0}" не является правильной для ...
Tabs_PropertySetBeforeInitialization	{0} не может быть изменен перед инициализац...
ReorderList_DropWatcherBehavior_...	Невозможно найти ребенка листа с id "{0}"
CascadingDropDown_MethodTimeout	[Превышено время ожидания в методе]
ExtenderBase_PageNotRegisteredFo...	Этот Page не был зарегистрирован для обратны...
Animation_NoDynamicPropertyFound	AjaxControlToolkit.Animation.createAnimation не н...
Animation_InvalidBaseType	AjaxControlToolkit.Animation.registerAnimation мо...
ResizableControlBehavior_InvalidHa...	{0} обработчик не является функцией, именем ф...
Animation_InvalidColor	Цвето должен быть в 7-ми символьном шестнад...
PasswordStrength_RemainingMixed...	Требуются символ(ы) в разных регистрах
CascadingDropDown_NoParentElem...	Невозможно найти родителя элемента "{0}"
ValidatorCallout_DefaultErrorMessage	Данный элемент не является валидным
ReorderList_DropWatcherBehavior_...	Ошибка пересортировки, дополнительная инфо...
PopupControl_NoDefaultProperty	Свойство по умолчанию типа "{1}" для control "{0}...
PopupExtender_NoParentElement	Невозможно найти родителя элемента "{0}"
TextCount_DefaultOverwriteText	Презаписать
PasswordStrength_RemainingNumb...	Еще {0} цифр(ы)
ResizableControlBehavior_CannotCh...	Изменения для {0} не поддерживаются
TextCount_DefaultDisplayFormat	Кол-во: {0} Осталось букв: {1} Максимальная дли...
Common_InvalidPaddingUnit	Единица типа "{0}" не является правильной для ...
ExtenderBase_ControlNotRegistered...	Этот Control не был зарегистрирован для обратн...
Calendar_Today	Сегодня: {0}
Common_DateTime_InvalidFormat	Неверный формат

Figure 20-8

In this example, you can see that the ASP.NET AJAX control is registered on the page using the `@Register` directive. This directive points to the `AJAXControlToolkit` assembly and gives all controls that use this assembly reference a tag prefix of `cc1`, which is why you see the AlwaysVisibleControl-Extender control prefixed with a `<cc1:[control name]>`.

ASP.NET AJAX Control Toolkit Extenders

The first set of items you look at includes the new extenders that are part of the ASP.NET AJAX Control Toolkit. Extenders are basically controls that reach out and extend other controls. For an example of this, you can think of the ASP.NET Validation Controls (covered in Chapter 4 of this book) as extender controls themselves. For instance, you can add a `RequiredFieldValidator` server control to a page and associate it to a TextBox control. This extends the TextBox control and changes its behavior. Normally it would just accept text. Now, if nothing is entered into the control, then the control will trigger an event back to the RequiredFieldValidator control whose client-side behavior is controlled by JavaScript.

The ASP.NET AJAX Control Toolkit's extenders pretty much accomplish the same thing. The controls extend the behavior of the ASP.NET server controls with additional JavaScript on the client as well as some server-side communications.

The ASP.NET AJAX extender controls are built using the ASP.NET AJAX extensions framework. The next few pages focus on using these new extenders within your ASP.NET applications.

AlwaysVisibleControlExtender

The AlwaysVisibleControlExtender allows you to specify controls that will always be present on a long page that requires scrolling. With this control, you can specify a location of the page where the control will always be present, no matter how the end user scrolls the page. Your options are to assign the vertical or horizontal alignment of the control that will always be visible. An example of using the AlwaysVisibleControlExtender is presented in Listing 20-2.

Listing 20-2: Using the AlwaysVisibleControlExtender

VB

```
<%@ Page Language="VB" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Response.Write("The page has been submitted!")
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>AlwaysVisibleControlExtender</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server" />
        <cc1:AlwaysVisibleControlExtender ID="AlwaysVisibleControlExtender1"
```

Continued

```
             runat="server" TargetControlID="Panel1" HorizontalOffset="10"
             HorizontalSide="Right" VerticalOffset="10">
         </cc1:AlwaysVisibleControlExtender>
         Form Element :
         <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
         <br />
         Form Element :
         <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
         <br />

         <!-- Excessive code removed for clarity -->

         Form Element :
         <asp:TextBox ID="TextBox29" runat="server"></asp:TextBox>
         <br />
         Form Element :
         <asp:TextBox ID="TextBox30" runat="server"></asp:TextBox>
         <br />
         <br />
         <asp:Panel ID="Panel1" runat="server">
             <asp:Button ID="Button1" runat="server" Text="Submit"
              OnClick="Button1_Click" />
             <asp:Button ID="Button2" runat="server" Text="Clear" />
         </asp:Panel>
     </div>
     </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Response.Write("The page has been submitted!");
    }
</script>
```

This code presents a very long form that requires end users to scroll the page in their browser. The AlwaysVisibleControlExtender control is present and its presence requires that you also have a ScriptManager control on the page (this is the same requirement for all ASP.NET AJAX controls).

The AlwaysVisibleControlExtender1 control extends the `Panel1` control through the use of the `TargetControlID` attribute. In this case, the value of the `TargetControlID` attribute points to the Panel1 control. The Panel1 control contains the form's Submit button. The result of the code from Listing 20-2 is shown in Figure 20-9.

The location of the Submit and Clear buttons on the page is controlled via a combination of a several control attributes. First off, the location on the page is determined by the `HorizontalSide` (possible values include `Center`, `Left`, and `Right`) and `VerticalSide` properties (possible values include `Bottom`, `Middle`, and `Top`). Then a padding is placed around the control using the `HorizontalOffset` and `VerticalOffset` properties, both of which are set to `10` pixels in this example.

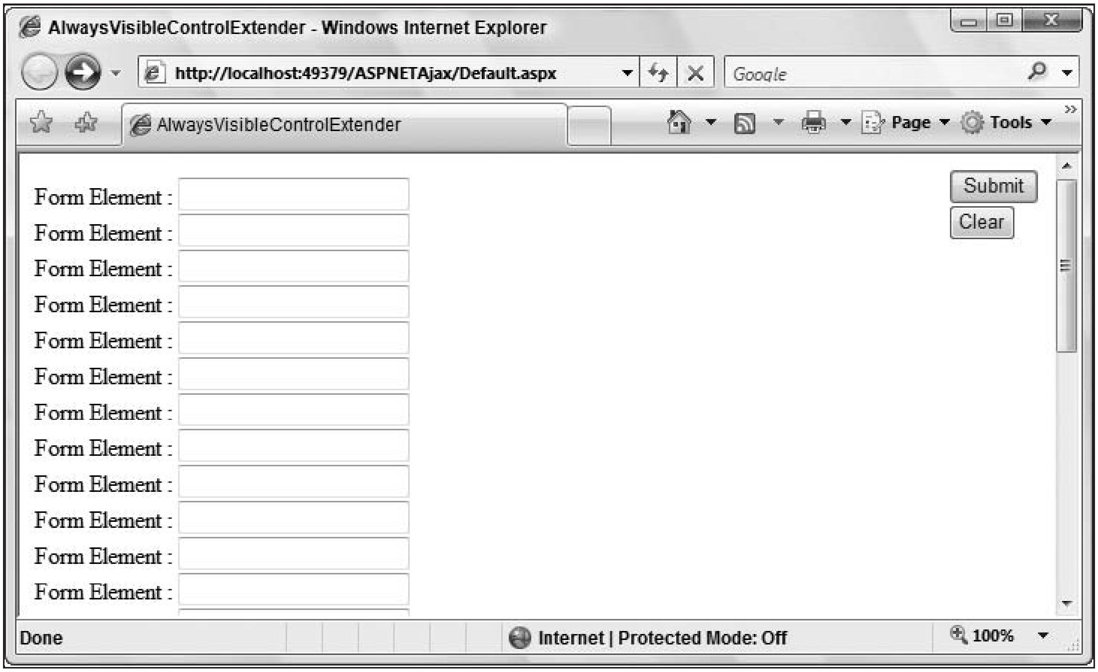

AlwaysVisibleControlExtender - Windows Internet Explorer
http://localhost:49379/ASPNETAjax/Default.aspx
Google
AlwaysVisibleControlExtender
Page
Tools
Form Element :
Form Element :
Form Element :
Form Element :
Form Element :
Form Element :
Form Element :
Form Element :
Form Element :
Form Element :
Form Element :
Form Element :
Submit
Clear
Done
Internet | Protected Mode: Off
100%

Figure 20-9

AnimationExtender

The AnimationExtender server control provides a tremendous amount of capabilities. It allows you to program fluid animations to the controls that you put on the page. There is a lot that you can do with this control (much more than can be shown in this chapter).

This control allows you to program elements that can move around the page based upon specific end user triggers (such as a button click). There are specific events available for you to program your animations against. These events are as follows:

- `OnClick`
- `OnHoverOver`
- `OnHoverOut`
- `OnLoad`
- `OnMouseOver`
- `OnMouseOut`

Creating animations is not as straightforward as many would like because there is little Visual Studio support (such as wizards or even IntelliSense). For an example of creating your first animation, Listing 20-3 shows how you can fade an element in and out of the page based upon an end user action.

Listing 20-3: Using the AnimationExtender to fade a background color

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>AnimationExtender</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server" />
        <cc1:AnimationExtender ID="AnimationExtender1" runat="server"
            TargetControlID="Panel1">
            <Animations>
                <OnClick>
                    <Sequence>
                        <Color PropertyKey="background" StartValue="#999966"
                         EndValue="#FFFFFF" Duration="5.0" />
                    </Sequence>
                </OnClick>
            </Animations>
        </cc1:AnimationExtender>
        <asp:Panel ID="Panel1" runat="server" BorderColor="Black"
            BorderWidth="3px" Font-Bold="True" Width="600px">
            Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
            Donec accumsan lorem. Ut consectetuer tempus metus. Aenean tincidunt
            venenatis tellus. Suspendisse molestie cursus ipsum. Curabitur ut
            lectus. Nulla ac dolor nec elit convallis vulputate. Nullam pharetra
            pulvinar nunc. Duis orci. Phasellus a tortor at nunc mattis congue.
            Vestibulum porta tellus eu orci. Suspendisse quis massa. Maecenas
            varius, erat non ullamcorper nonummy, mauris erat eleifend odio, ut
            gravida nisl neque a ipsum. Vivamus facilisis. Cras viverra. Curabitur
            ut augue eget dolor semper posuere. Aenean at magna eu eros tempor
            pharetra. Aenean mauris.
        </asp:Panel>
    </div>
    </form>
</body>
</html>
```

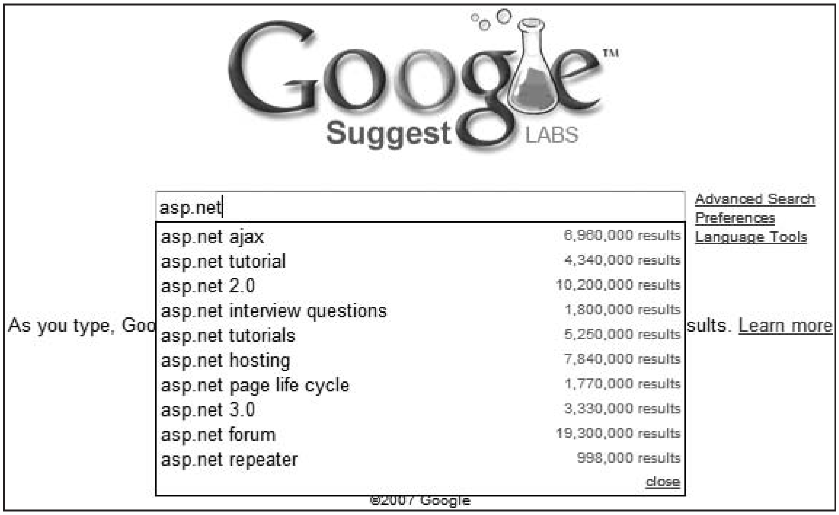
In this case, when you pull up the page from Listing 20-3, you will see that it uses a single AnimationExtender control that is working off the `Panel1` control. This connection is made using the `TargetControlID` property.

As stated, Intellisense is not enabled when you are typing the code that is contained within the AnimationExtender control, so you are going to have to look in the documentation for the animations that you want to create. In the case of the previous example, the `<OnClick>` element is utilized to define a sequence of events that need to occur when the control is clicked. For this example, there is only one animation defined within the `<Sequence>` element — a color change to the background of the element. Here, the `<Color>` element states that the background CSS property will need to start at the color `#999966` and change completely to color `#FFFFFF` within 5 seconds (defined using the `Duration` property).

When you pull up this page and click on the Panel element, you will see the color change in a five-second duration from the described start color to the end color.

AutoCompleteExtender

The `AutoCompleteExtender` control provides you the ability to help the end user find what they might be looking for when they have to type in search terms within a text box. Like the product Google Suggest (shown in Figure 20-10), once you start typing characters in the text box, you will get results from a datastore that matches what you have typed so far.



asp.net

asp.net ajax	6,960,000 results
asp.net tutorial	4,340,000 results
asp.net 2.0	10,200,000 results
asp.net interview questions	1,800,000 results
asp.net tutorials	5,250,000 results
asp.net hosting	7,840,000 results
asp.net page life cycle	1,770,000 results
asp.net 3.0	3,330,000 results
asp.net forum	19,300,000 results
asp.net repeater	998,000 results

Figure 20-10

To establish something similar for yourself, create a new page that contains only a ScriptManager control, an AutoCompleteExtender control, and a TextBox control. The ASP.NET portion of the page should appear as presented in Listing 20-4.

Listing 20-4: The ASP.NET page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="AutoComplete.aspx.cs"
    Inherits="AutoComplete" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>
```

Continued

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>AutoComplete</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:AutoCompleteExtender ID="AutoCompleteExtender1" runat="server"
            TargetControlID="TextBox1" ServiceMethod="GetCompletionList"
            UseContextKey="True">
        </cc1:AutoCompleteExtender>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </div>
    </form>
</body>
</html>
```

Again, like the other ASP.NET AJAX controls, you extend the TextBox control using the `TargetControlID` property. When you first add these controls to the page, you will not have the `ServiceMethod` property defined in the AutoCompleteExtender control. Using Visual Studio 2008, you can make the framework for a service method and tie the extender control to this method all from the design surface. After expanding the TextBox control's smart tag, select the Add AutoComplete page method option from the provided menu, shown in Figure 20-11.

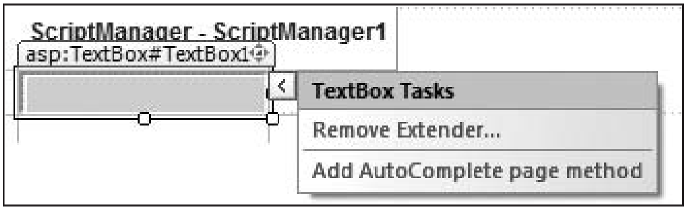


Figure 20-11

This action creates a service method in the code-behind for your page. Listing 20-4 shows the steps necessary to complete this method to call the company names from the Northwind database.

Instructions on downloading and using the Northwind database can be found in Chapter 8.

Listing 20-4: The code-behind that sets up the service method for auto-complete

VB

```
Imports System.Data
Imports System.Data.SqlClient

Partial Class AutoComplete
    Inherits System.Web.UI.Page
    <System.Web.Services.WebMethodAttribute(), _
     System.Web.Script.Services.ScriptMethodAttribute()> _
    Public Shared Function GetCompletionList(ByVal prefixText As String, _
      ByVal count As Integer) As String()
        Dim conn As SqlConnection
        Dim cmd As SqlCommand
        Dim cmdString As String =
           "Select CompanyName from Customers WHERE CompanyName LIKE '" & _
           prefixText & "%'"
        conn = New SqlConnection("Data Source=.\SQLEXPRESS;
           AttachDbFilename=|DataDirectory|\NORTHWND.MDF;
           Integrated Security=True;User Instance=True")
        ' Put this string on one line in your code
        cmd = New SqlCommand(cmdString, conn)
        conn.Open()

        Dim myReader As SqlDataReader
        Dim returnData As List(Of String) = New List(Of String)
        myReader = cmd.ExecuteReader(CommandBehavior.CloseConnection)

        While myReader.Read()
            returnData.Add(myReader("CompanyName").ToString())
        End While

        Return returnData.ToArray()
    End Function
End Class
```

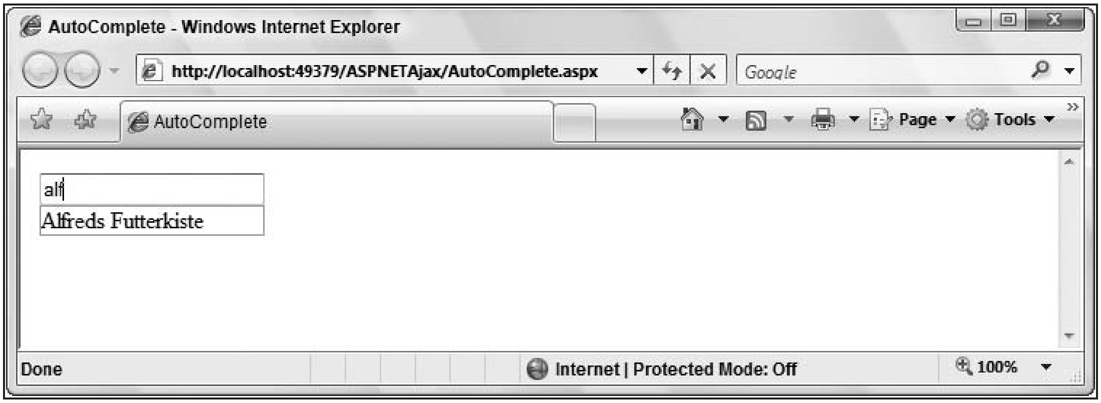
C#

```
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;

public partial class AutoComplete : System.Web.UI.Page
{
    [System.Web.Services.WebMethodAttribute(),
     System.Web.Script.Services.ScriptMethodAttribute()]
    public static string[] GetCompletionList(string prefixText, int count,
       string contextKey)
    {
        SqlConnection conn;
        SqlCommand cmd;
        string cmdString =
           "Select CompanyName from Customers WHERE CompanyName LIKE '" +
           prefixText + "%'";
        conn = new
            SqlConnection(@"Data Source=.\SQLEXPRESS;
            AttachDbFilename=|DataDirectory|\NORTHWND.MDF;
            Integrated Security=True;User Instance=True");
        // Put this string on one line in your code
        cmd = new SqlCommand(cmdString, conn);
        conn.Open();

        SqlDataReader myReader;
```

Continued

```
            List<string> returnData = new List<string>();

            myReader = cmd.ExecuteReader(CommandBehavior.CloseConnection);

            while (myReader.Read())
            {
                returnData.Add(myReader["CompanyName"].ToString());
            }

            return returnData.ToArray();
        }
    }
```

When you run this page and type the characters **alf** into the text box, the `GetCompletionList()` method is called, passing in these characters. These characters are retrievable through the `prefixText` parameter (you can also use the `count` parameter, which is defaulted at `10`). The Northwind database is called using the `prefixText` value and this is what is returned back to the TextBox1 control. In the end, you get a drop-down list of the items that match the first three characters that were entered into the text box. This is illustrated in Figure 20-12.


AutoComplete - Windows Internet Explorer
http://localhost:49379/ASPNETAjax/AutoComplete.aspx
Google
AutoComplete
Page
Tools
Done
Internet | Protected Mode: Off
100%

alf

Alfreds Futterkiste

Figure 20-12

It is good to know that the results, once called the first time, are cached. This caching is controlled via the `EnableCaching` property (it is defaulted to `true`). You can also change the style of the drop-down auto-complete list, configure how many elements appear, and many more points of this feature. One more important point is that you are not required to call a method that is exposed out on the same page as the control as the example in this book demonstrates, but you can also call another server-side method on another page, or a Web method.

CalendarExtender

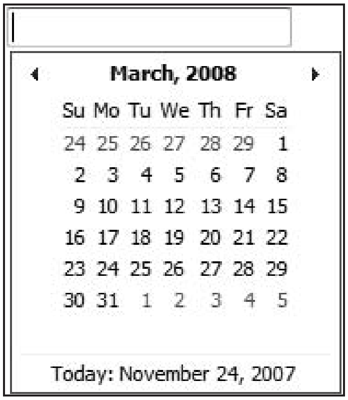
If there is one problem with a form that slows form submission up, it is selecting dates and trying to figure out which format of the date the form requires. The CalendarExtender control is a solution that makes it simple for your end users to select a date within a form.

The quickest way for end users to select a date in a form is to have a calendar that they can navigate to find the date they require. The calendar date can then be translated to a textual date format in the text box. The CalendarExtender control gives you all the client-side code required for this kind of action. Listing 20-5 shows you an example of providing a calendar control off your textbox controls.

Listing 20-5: Using a calendar control from a TextBox control

```
<%@ Page Language="VB" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CalendarExtender</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:CalendarExtender ID="CalendarExtender1" runat="server"
         TargetControlID="TextBox1">
        </cc1:CalendarExtender>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </div>
    </form>
</body>
</html>
```

When you run this page, the result will be a single text box on the page and the text box will appear no different from any other text box. However, when the end user clicks inside of the text box, a calendar appears directly below it as shown in Figure 20-13.



March, 2008

Su	Mo	Tu	We	Th	Fr	Sa
24	25	26	27	28	29	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Today: November 24, 2007

Figure 20-13

Then, when the end user selects a date from the calendar, the date is placed as text within the text box as illustrated in Figure 20-14.

3/8/2008

Figure 20-14

Some of the properties exposed from this control are `FirstDayOfWeek` and `PopupPosition` (which has the options `BottomLeft`, `BottomRight`, `TopLeft`, and `TopRight`). You can also change how the calendar is initiated on the client. Some sites offer a calendar button next to the text box and only popup the calendar option when the end user clicks the button. If this is something that you would like to do on your pages, then you should use the `PopupButtonID` property, which you must point to the ID of the image or button that you are using.

CollapsiblePanelExtender

The CollapsiblePanelExtender server control allows you to collapse one control into another. When working with two Panel server controls, you can provide a nice way to control any real estate issues that you might be experiencing on your ASP.NET page.

An example of using the CollapsiblePanelExtender control with two Panel controls is illustrated here in Listing 20-6.

Listing 20-6: Using CollapsiblePanelExtender with two Panel controls

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>CollapsiblePanelExtender</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:CollapsiblePanelExtender ID="CollapsiblePanelExtender1" runat="server"
         TargetControlID="Panel2" Collapsed="True" ExpandControlID="Panel1"
         CollapseControlID="Panel2">
        </cc1:CollapsiblePanelExtender>
        <asp:Panel ID="Panel1" runat="server" BackColor="#000066"
         ForeColor="White">
            <asp:Label ID="Label2" runat="server"
             Text="This is my title"></asp:Label>
            <asp:Label ID="Label1" runat="server"
             Text="[Click to expand or collapse]"></asp:Label>
        </asp:Panel>
        <asp:Panel ID="Panel2" runat="server">
            Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
            Donec accumsan lorem. Ut consectetuer tempus metus. Aenean tincidunt
            venenatis tellus. Suspendisse molestie cursus ipsum. Curabitur ut
            lectus. Nulla ac dolor nec elit convallis vulputate. Nullam pharetra
            pulvinar nunc. Duis orci. Phasellus a tortor at nunc mattis congue.
            Vestibulum porta tellus eu orci. Suspendisse quis massa. Maecenas
            varius, erat non ullamcorper nonummy, mauris erat eleifend odio, ut
            gravida nisl neque a ipsum. Vivamus facilisis. Cras viverra. Curabitur
            ut augue eget dolor semper posuere. Aenean at magna eu eros tempor
            pharetra. Aenean mauris.
        </asp:Panel>
    </div>
    </form>
</body>
</html>
```

In this case, when the page is pulled up for the first time you will only see the contents of `Panel1`–the title panel. By default, you would usually see both controls, but since the `Collapsed` property is set to `True` in the control, you will only see Panel1. Clicking the Panel control will then expose the contents of Panel2. In fact, the contents will slide out from the Panel1 control. Tying these two controls together to do this action is accomplished through the use of the CollapsiblePanelExtender control. This control's `TargetControlID` is assigned to the second Panel control — Panel2, as this is the control that needs to expand onto the page. The `ExpandControlID` property is the control that initiates the expansion.

Once expanded, it is when the end user clicks on the Panel2 that the contents will disappear by sliding back into Panel1. This is accomplished through the use of the `CollapseControlID` property being assigned to `Panel2`.

The CollapsiblePanelExtender control has a number of properties that allow you to fine-tune how the expanding and collapsing occur. For instance, you could have also set the Label1 control to be the initiator of this process and even change the text of the Label control depending on the whether the Panel2 is collapsed or expanded. This use is illustrated in Listing 20-7.

Listing 20-7: Using a Label control to expand or collapse the Panel control

```
<cc1:CollapsiblePanelExtender ID="CollapsiblePanelExtender1" runat="server"
 TargetControlID="Panel2" Collapsed="True" ExpandControlID="Label1"
 CollapseControlID="Label1"
 CollapsedText="[Click to expand]"
 ExpandedText="[Click to collapse]"
 TextLabelID="Label1">
</cc1:CollapsiblePanelExtender>
```

In this case, when the end user clicks on the Label1 control, not only will Panel2 expand and collapse, but the text within the Label1 control will change accordingly.

ConfirmButtonExtender and ModalPopupExtender

Usually before allowing your end users to make deletions of data via a browser application, you want to confirm with the end user upon such actions. ConfirmButtonExtender allows you to question the end user's action and reconfirm that they want the action to occur. Listing 20-8 shows how to use this control.

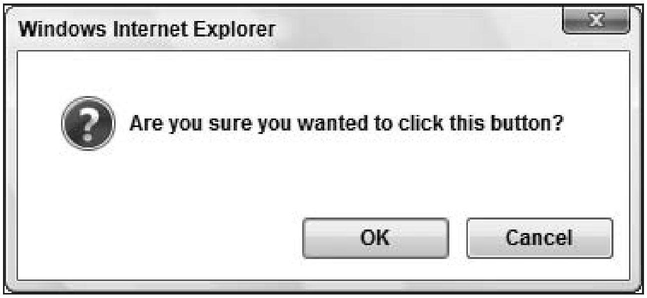
Listing 20-8: Using the ConfirmButtonExtender control to reconfirm a user action

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>
```

Continued

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>ConfirmButtonExtender</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:ConfirmButtonExtender ID="ConfirmButtonExtender1" runat="server"
         TargetControlID="Button1"
         ConfirmText="Are you sure you wanted to click this button?">
        </cc1:ConfirmButtonExtender>
        <asp:Button ID="Button1" runat="server" Text="Button" />
    </div>
    </form>
</body>
</html>
```

In this case, the ConfirmButtonExtender extends the Button1 server control and adds a confirmation dialog using the text defined with the `CofirmText` property. This page is shown in Figure 20-15.



Are you sure you wanted to click this button?

Figure 20-15

If the end user clicks OK in this instance, then the page will function normally as if the dialog never occurred. However, if Cancel is clicked, by default the dialog will disappear and the form will not be submitted (it will be as if the button was not clicked at all). In this case, you can also capture the Cancel button being clicked and perform a client-side operation by using the `OnClientClick` event and giving it a value of a client-side JavaScript function.

Instead of using the browsers modal dialogs, you can even go as far as creating your own to use as the confirmation form. To accomplish this task, you will need to use the new ModalPopupExtender server control. The `ModalPopupExtender` control points to another control to use for the confirmation. Listing 20-9 shows how to make use of this control.

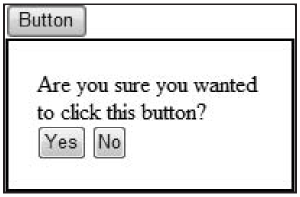
Listing 20-9: Using the ModalPopupExtender control to create your own confirmation form

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
 TagPrefix="cc1" %>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>ConfirmButtonExtender</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:ConfirmButtonExtender ID="ConfirmButtonExtender1" runat="server"
         TargetControlID="Button1"
         DisplayModalPopupID="ModalPopupExtender1">
        </cc1:ConfirmButtonExtender>
        <cc1:ModalPopupExtender ID="ModalPopupExtender1" runat="server"
            CancelControlID="ButtonNo" OkControlID="ButtonYes"
            PopupControlID="Panel1"
            TargetControlID="Button1">
        </cc1:ModalPopupExtender>
        <asp:Button ID="Button1" runat="server" Text="Button" />
        <asp:Panel ID="Panel1" runat="server"
         style="display:none; background-color:White; width:200;
         border-width:2px; border-color:Black; border-style:solid; padding:20px;">
         Are you sure you wanted to click this button?<br />
         <asp:Button ID="ButtonYes" runat="server" Text="Yes" />
         <asp:Button ID="ButtonNo" runat="server" Text="No" />
        </asp:Panel>
    </div>
    </form>
</body>
</html>
```

In this example, the ConfirmButtonExtender still points to the Button1 control on the page, meaning that when the button is clicked, then the ConfirmButtonExtender will take action. Instead of using the `ConfirmText` property, the `DisplayModalPopupID` property is used. In this case, it points to the ModalPopupExtender1 control — another extender control.

The ModalPopupExtender control, in turn, references the Panel1 control on the page through the use of the `PopupControlID` property. The contents of this Panel control is used for the confirmation on the button click. For this to work, the ModalPopupExtender control has to have a value for the `OkControlID` and the `CancelControlID` properties. In this case, these two properties point to the two Button controls that are contained within the Panel control. When you run this page, you will get the results shown in Figure 20-16.



Are you sure you wanted to click this button?

Figure 20-16

DragPanelExtender

The DragPanelExtender enables you define areas where end users can move elements around the page as they wish. The end user actually has the ability to drag and drop the element anywhere on the browser page.

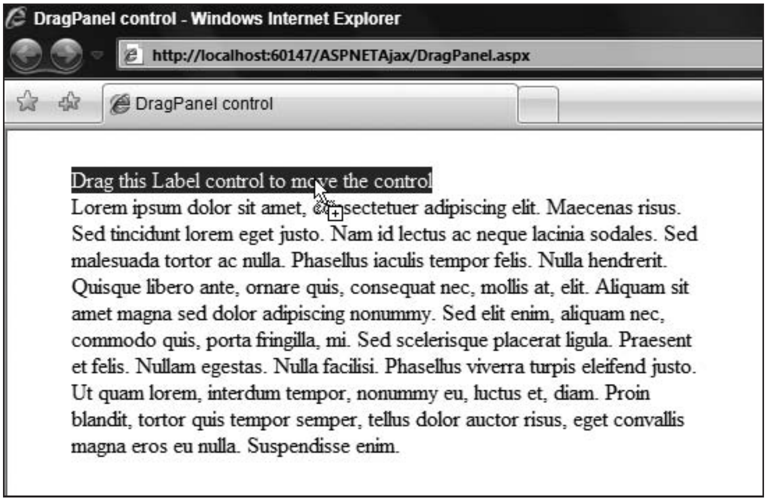
To enable this feature, you have to do a few things. The first suggestion is to create a `<div>` area on the page that is large enough for to drag the item around in. From here, you need to specify what will be used as the drag handle and another control that will follow the drag handle around. In the example in Listing 20-10, the Label control is used as the drag handle, and the Panel2 control is the content that is dragged around the screen.

Listing 20-10: Dragging a Panel control around the page

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>DragPanel control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <div style="height: 600px;">
            <cc1:DragPanelExtender ID="DragPanelExtender1" runat="server"
             DragHandleID="Label1" TargetControlID="Panel1">
            </cc1:DragPanelExtender>
            <asp:Panel ID="Panel1" runat="server" Width="450px">
                <asp:Label ID="Label1" runat="server"
                 Text="Drag this Label control to move the control"
                 BackColor="DarkBlue" ForeColor="White"></asp:Label>
                <asp:Panel ID="Panel2" runat="server" Width="450px">
            Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
            Donec accumsan lorem. Ut consectetuer tempus metus. Aenean tincidunt
            venenatis tellus. Suspendisse molestie cursus ipsum. Curabitur ut
            lectus. Nulla ac dolor nec elit convallis vulputate. Nullam pharetra
            pulvinar nunc. Duis orci. Phasellus a tortor at nunc mattis congue.
            Vestibulum porta tellus eu orci. Suspendisse quis massa. Maecenas
            varius, erat non ullamcorper nonummy, mauris erat eleifend odio, ut
            gravida nisl neque a ipsum. Vivamus facilisis. Cras viverra. Curabitur
            ut augue eget dolor semper posuere. Aenean at magna eu eros tempor
            pharetra. Aenean mauris.
                </asp:Panel>
            </asp:Panel>
        </div>
    </div>
    </form>
</body>
</html>
```

This example creates a `<div>` element that has a height of `600` pixels. Within this defined area, the example uses a DragPanelExtender control and targets the Panel1 control through the use of the `TargetControlID` property being assigned to this control.

Within the Panel1 control are two other server controls — a Label and another Panel control. The Label control is assigned to be the drag handle using the `DragHandleID` property of the DragPanelExtender control. With this little bit of code in place, you are now able to drag the `Panel1` control around on your browser window. Figure 20-17 shows the Label control being used as a handle to drag around the Panel control.


DragPanel control - Windows Internet Explorer
http://localhost:60147/ASPNETAjax/DragPanel.aspx
DragPanel control

Drag this Label control to move the control

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas risus. Sed tincidunt lorem eget justo. Nam id lectus ac neque lacinia sodales. Sed malesuada tortor ac nulla. Phasellus iaculis tempor felis. Nulla hendrerit. Quisque libero ante, ornare quis, consequat nec, mollis at, elit. Aliquam sit amet magna sed dolor adipiscing nonummy. Sed elit enim, aliquam nec, commodo quis, porta fringilla, mi. Sed scelerisque placerat ligula. Praesent et felis. Nullam egestas. Nulla facilisi. Phasellus viverra turpis eleifend justo. Ut quam lorem, interdum tempor, nonummy eu, luctus et, diam. Proin blandit, tortor quis tempor semper, tellus dolor auctor risus, eget convallis magna eros eu nulla. Suspendisse enim.

Figure 20-17

DropDownExtender

The DropDownExtender control allows you to take any control and provide a drop-down list of options below it for selection. It provides a different framework than a typical dropdown list control as it allows for an extreme level of customization. Listing 20-11 shows how you can even use an image as the initiator of drop-down list of options.

Listing 20-11: Using an Image control as an initiator of a drop-down list

VB

```
<%@ Page Language="VB" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
       Image1.ImageUrl = "Images/Creek.jpg"
    End Sub
```

Continued

```
    Protected Sub Option_Click(ByVal sender As Object, ByVal e As System.EventArgs)
       Image1.ImageUrl = "Images/" & DirectCast(sender, LinkButton).Text & ".jpg"
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>DropDownExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <cc1:DropDownExtender ID="DropDownExtender1" runat="server"
                 DropDownControlID="Panel1" TargetControlID="Image1">
                </cc1:DropDownExtender>
                <asp:Image ID="Image1" runat="server">
                </asp:Image>
            <asp:Panel ID="Panel1" runat="server" Height="50px" Width="125px">
                <asp:LinkButton ID="Option1" runat="server"
                 OnClick="Option_Click">Creek</asp:LinkButton>
                <asp:LinkButton ID="Option2" runat="server"
                 OnClick="Option_Click">Dock</asp:LinkButton>
                <asp:LinkButton ID="Option3" runat="server"
                 OnClick="Option_Click">Garden</asp:LinkButton>
            </asp:Panel>
            </ContentTemplate>
        </asp:UpdatePanel>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Image1.ImageUrl = "Images/Creek.jpg";
    }

    protected void Option_Click(object sender, EventArgs e)
    {
        Image1.ImageUrl = "Images/" + ((LinkButton)sender).Text + ".jpg";
    }
</script>
```

In this case, a DropDownExtender control is tied to an Image control that on the `Page_Load()` event displays a specific image. The DropDownExtender control has two specific properties that need to be filled. The first is the `TargetControlID` property that defines the control that becomes the initiator of the drop-down list. The second property is the `DropDownControlID` property, which defines the element on the page that will be used for the drop-down items that appear below the control. In this case, it is a Panel control with three LinkButton controls.

Each of the LinkButton controls designates a specific image that should appear on the page. Selecting one of the options changes the image to the choice through the `Option_Click()` method. Running this page gives you the results illustrated in Figure 20-18.

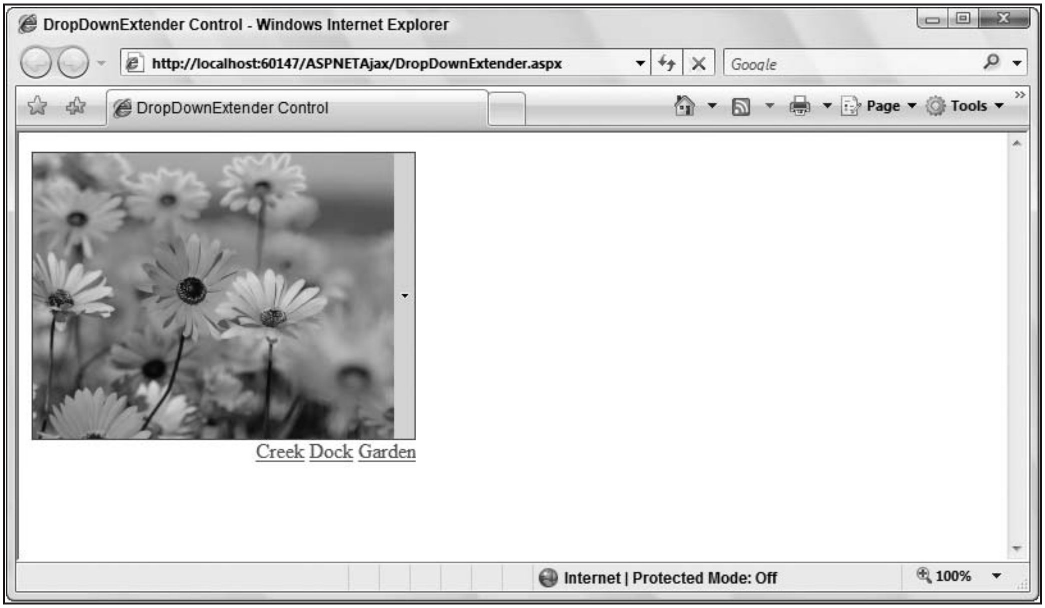

DropDownExtender Control - Windows Internet Explorer
http://localhost:60147/ASPNETAjax/DropDownExtender.aspx
Google
DropDownExtender Control
Page
Tools
Creek Dock Garden
Internet | Protected Mode: Off
100%

Figure 20-18

DropShadowExtender

The DropShadowExtender control allows you to put a drop shadow on any control you choose as the target. The first thought is an image (as shown here in Listing 20-12), but you can use it for any control that you wish.

Listing 20-12: Using DropShadowExtender with an Image control

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

Continued

```
        <title>DropShadowExtender Control</title>
    </head>
    <body>
        <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <cc1:DropShadowExtender ID="DropShadowExtender1" runat="server"
             TargetControlID="Image1">
            </cc1:DropShadowExtender>
            <asp:Image ID="Image1" runat="server" ImageUrl="Images/Garden.jpg" />
        </div>
        </form>
    </body>
    </html>
```

In this example, it is as simple as using the DropShadowExtender control with a `TargetControlID` of `Image1`. With this in place, the image will appear in the browser as shown in Figure 20-19.

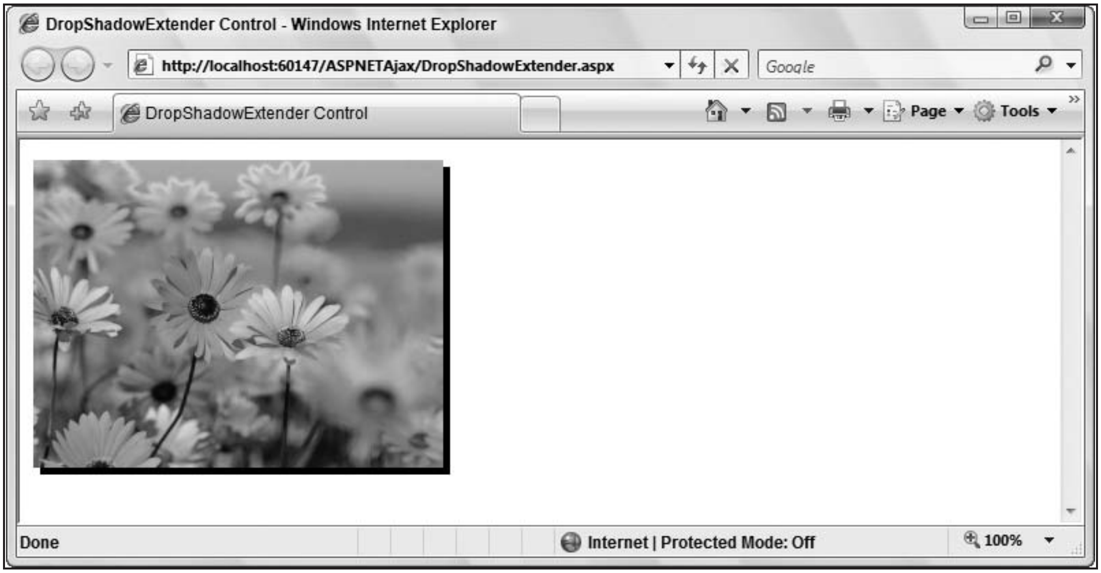

DropShadowExtender Control - Windows Internet Explorer
http://localhost:60147/ASPNETAjax/DropShadowExtender.aspx
Google
DropShadowExtender Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

Figure 20-19

As stated, in addition to images, you can use DropShadowExtender for almost anything. Listing 20-13 shows how to use it with a Panel control.

Listing 20-13: Using the DropShadowExtender with a Panel control

```
    <%@ Page Language="C#" %>

    <%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
        TagPrefix="cc1" %>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>DropShadowExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:DropShadowExtender ID="DropShadowExtender1" runat="server"
         TargetControlID="Panel1" Rounded="True">
        </cc1:DropShadowExtender>
        <asp:Panel ID="Panel1" runat="server" BackColor="Orange" Width="300"
         HorizontalAlign="Center">
            <asp:Login ID="Login1" runat="server">
            </asp:Login>
        </asp:Panel>
    </div>
    </form>
</body>
</html>
```

In this case, a Panel control with a Login control is extended with the DropShadowExtender control. The result is quite similar to that of the Image control's result. However, one addition to the DropShadowExtender control here is that the `Rounded` property is set to `True` (by default, it is set to `False`). This produces the look shown in Figure 20-20.

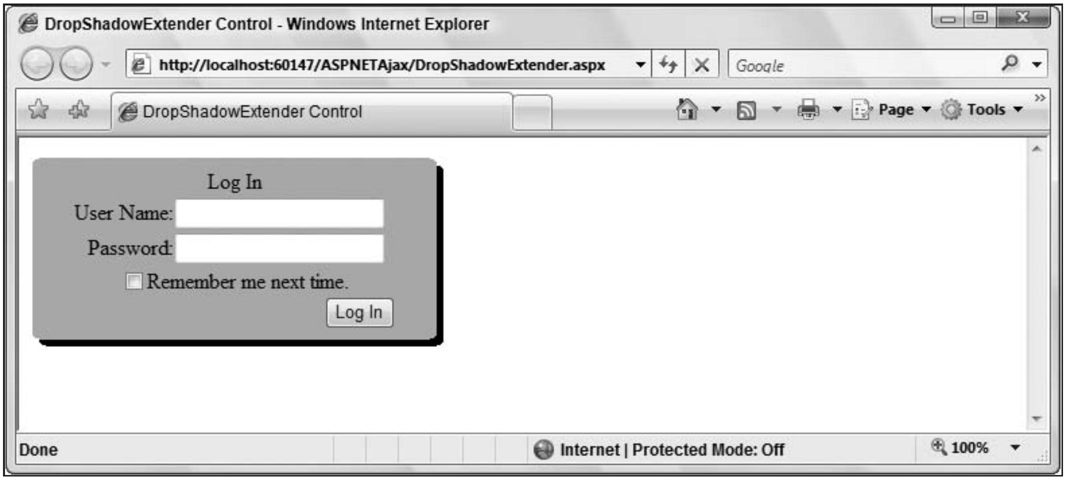

DropShadowExtender Control - Windows Internet Explorer
http://localhost:60147/ASPNETAjax/DropShadowExtender.aspx
Google
DropShadowExtender Control
Page
Tools
Log In
User Name:
Password:
Remember me next time.
Log In
Done
Internet | Protected Mode: Off
100%

Figure 20-20

As you can see from Figure 20-20, not only are the edges of the drop shadow rounded, but also the entire Panel control has rounded edges. Other style properties that you can work with include the `Opacity` property, which controls the opacity of the drop shadow only, and the `Radius` property, which controls the radius used in rounding the edges and obviously works only if the `Rounded` property is set to `True`. By default, the `Opacity` setting is set at `1`, which means 100% visible. To set it at, say, 50% opacity, you have to set the `Opacity` value to `.5`.

DynamicPopulateExtender

The DynamicPopulateExtender control allows you to send dynamic HTML output to a Panel control. For this to work, you need one control or event that triggers a call back to the server to get the HTML that in turn gets pushed into the `Panel` control, thereby making a dynamic change on the client.

As with the AutoCompleteExtender control, you need a server-side event that returns something back to the client asynchronously. Listing 20-14 shows the code required to use this control on the `.aspx` page.

Listing 20-14: Using the DynamicPopulateExtender control to populate a Panel control

.ASPX

```
<%@ Page Language="VB" AutoEventWireup="true"
    CodeFile="DynamicPopulateExtender.aspx.vb"
    Inherits="DynamicPopulateExtender" %>

<%@ Register Assembly="AjaxControlToolkit, Version=3.5.11119.20050,
    Culture=neutral, PublicKeyToken=28f01b0e84b6d53e"
    Namespace="AjaxControlToolkit" TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>DynamicPopulateExtender Control</title>
    <script type="text/javascript">
      function updateGrid(value) {
        var behavior = $find('DynamicPopulateExtender1');
        if (behavior) {
            behavior.populate(value);
        }
      }
    </script>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server" />
        <cc1:DynamicPopulateExtender ID="DynamicPopulateExtender1" runat="server"
            TargetControlID="Panel1" ServiceMethod="GetDynamicContent">
        </cc1:DynamicPopulateExtender>
        <div onclick="updateGrid(this.value);" value='0'>
        <asp:LinkButton ID="LinkButton1" runat="server"
         OnClientClick="return false;">Customers</asp:LinkButton></div>
        <div onclick="updateGrid(this.value);" value='1'>
        <asp:LinkButton ID="LinkButton2" runat="server"
         OnClientClick="return false;">Employees</asp:LinkButton></div>
        <div onclick="updateGrid(this.value);" value='2'>
        <asp:LinkButton ID="LinkButton3" runat="server"
         OnClientClick="return false;">Products</asp:LinkButton></div>
        <asp:Panel ID="Panel1" runat="server">
        </asp:Panel>
    </div>
    </form>
</body>
</html>
```

This `.aspx` page is doing a lot. First off, there is a client-side JavaScript function called `updateGrid()`. This function calls the DynamicPopulateExtender control that is on the page. You will also find three LinkButton server controls, each of which is encased within a `<div>` element that calls the `updateGrid()` function and provides a value that is passed into the function. Since you want the `<div>` element's `onclick` event to be triggered with a click and not the `LinkButton` control's click event, each LinkButton contains an `OnClientClick` attribute that simply does nothing. This is accomplished using `return false;`.

The DynamicPopulateExtender control on the page targets the Panel1 control as the container that will take the HTML that comes from the server on an asynchronous request. The DynamicPopulateExtender control knows where to go get the HTML using the `ServiceMethod` attribute. The value of this attribute calls the `GetDynamicContent()` method, which is in the page's code-behind file.

Once you have the `.aspx` page in place, the next step is to create the code-behind page. This page will contain the server-side method that is called by the DynamicPopulateExtender control. This is presented in Listing 20-15.

Listing 20-15: The code-behind page of the DynamicPopulateExtender.aspx page

VB

```
Imports System.Data
Imports System.Data.SqlClient
Imports System.IO

Partial Class DynamicPopulateExtender
    Inherits System.Web.UI.Page

    <System.Web.Services.WebMethodAttribute()> _
    <System.Web.Script.Services.ScriptMethodAttribute()> _
    Public Shared Function GetDynamicContent(ByVal contextKey As System.String) _
      As System.String
        Dim conn As SqlConnection
        Dim cmd As SqlCommand
        Dim cmdString As String = "Select * from Customers"

        Select Case contextKey
            Case "1"
                cmdString = "Select * from Employees"
            Case "2"
                cmdString = "Select * from Products"
        End Select

        conn = New SqlConnection("Data Source=.\SQLEXPRESS;
            AttachDbFilename=|DataDirectory|\NORTHWND.MDF;
            Integrated Security=True;User Instance=True")
            ' Put this string on one line in your code
        cmd = New SqlCommand(cmdString, conn)
        conn.Open()

        Dim myReader As SqlDataReader
        myReader = cmd.ExecuteReader(CommandBehavior.CloseConnection)
```

Continued

```
        Dim dt As New DataTable
        dt.Load(myReader)
        myReader.Close()

        Dim myGrid As New GridView
        myGrid.ID = "GridView1"
        myGrid.DataSource = dt
        myGrid.DataBind()

        Dim sw As New StringWriter
        Dim htw As HtmlTextWriter = New HtmlTextWriter(sw)

        myGrid.RenderControl(htw)
        htw.Close()

        Return sw.ToString()
    End Function
End Class
```

C#

```
using System.Data;
using System.Data.SqlClient;
using System.IO;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class DynamicPopulateExtender : System.Web.UI.Page
{
    [System.Web.Services.WebMethodAttribute(),
     System.Web.Script.Services.ScriptMethodAttribute()]
    public static string GetDynamicContent(string contextKey)
    {
        SqlConnection conn;
        SqlCommand cmd;
        string cmdString = "Select * from Customers";

        switch (contextKey)
        {
            case ("1"):
                cmdString = "Select * from Employees";
                break;
            case ("2"):
                cmdString = "Select * from Products";
                break;
        }

        conn = new
            SqlConnection(@"Data Source=.\SQLEXPRESS;
               AttachDbFilename=|DataDirectory|\NORTHWND.MDF;
               Integrated Security=True;User Instance=True");
               // Put this string on one line in your code
        cmd = new SqlCommand(cmdString, conn);
        conn.Open();
```

```
            SqlDataReader myReader;
            myReader = cmd.ExecuteReader(CommandBehavior.CloseConnection);

            DataTable dt = new DataTable();
            dt.Load(myReader);
            myReader.Close();

            GridView myGrid = new GridView();
            myGrid.ID = "GridView1";
            myGrid.DataSource = dt;
            myGrid.DataBind();

            StringWriter sw = new StringWriter();
            HtmlTextWriter htw = new HtmlTextWriter(sw);

            myGrid.RenderControl(htw);
            htw.Close();

            return sw.ToString();
        }
    }
```

This code is the code-behind page for the `DynamicPopulateExtender.aspx` page and contains a single method that is callable asynchronously. The `GetDynamicContent()` method takes a single parameter, `contextKey`, a string value that can be used to determine what link the end user clicked.

Based upon the selection, a specific command string is used to populate a `DataTable` object. From here, the `DataTable` object is used as the data source for a programmatic GridView control that is rendered and returned as a string to the client. The client will take the large string and use the text to populate the Panel1 control that is on the page. The result of clicking one of the links is shown in Figure 20-21.

FilteredTextBoxExtender

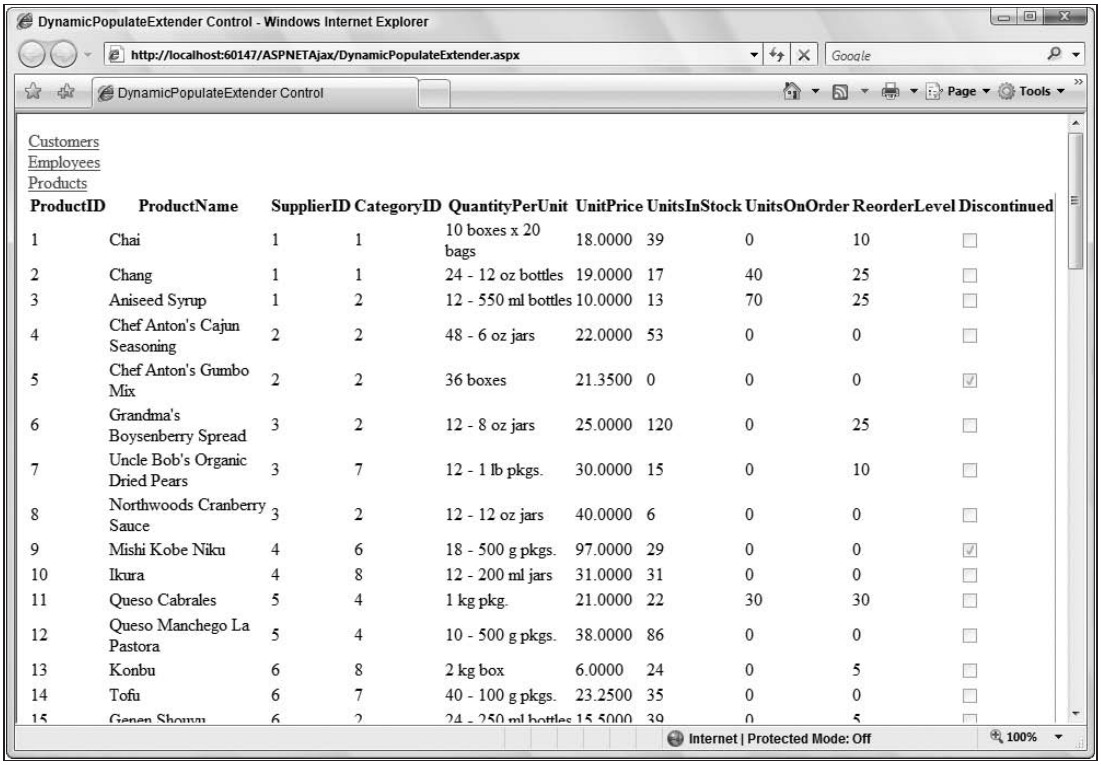
The FilteredTextBoxExtender control works off a `TextBox` control to specify the types of characters the end user can input into the control. For instance, if you want the end user to be able to enter only numbers into the text box, then you can associate a FilteredTextBoxExtender to the TextBox control and specify such behavior. An example of this is presented in Listing 20-16.

Listing 20-16: Filtering a text box to use only numbers

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>FilteredTextBoxExtender Control</title>
</head>
```

Continued

```
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:FilteredTextBoxExtender ID="FilteredTextBoxExtender1" runat="server"
         TargetControlID="TextBox1" FilterType="Numbers">
        </cc1:FilteredTextBoxExtender>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </div>
    </form>
</body>
</html>
```


DynamicPopulateExtender Control - Windows Internet Explorer
http://localhost:60147/ASPNETAjax/DynamicPopulateExtender.aspx
Google
DynamicPopulateExtender Control
Page
Tools
Internet | Protected Mode: Off
100%

Customers
Employees
Products

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
1	Chai	1	1	10 boxes x 20 bags	18.0000	39	0	10	☐
2	Chang	1	1	24 - 12 oz bottles	19.0000	17	40	25	☐
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10.0000	13	70	25	☐
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22.0000	53	0	0	☐
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.3500	0	0	0	☑
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25.0000	120	0	25	☐
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30.0000	15	0	10	☐
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40.0000	6	0	0	☐
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97.0000	29	0	0	☑
10	Ikura	4	8	12 - 200 ml jars	31.0000	31	0	0	☐
11	Queso Cabrales	5	4	1 kg pkg.	21.0000	22	30	30	☐
12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.	38.0000	86	0	0	☐
13	Konbu	6	8	2 kg box	6.0000	24	0	5	☐
14	Tofu	6	7	40 - 100 g pkgs.	23.2500	35	0	0	☐
15	Genen Shouyu	6	2	24 - 250 ml bottles	15.5000	39	0	5	☐

Figure 20-21

In this case, a FilteredTextBoxExtender control is attached to the TextBox1 control through the use of the `TargetControlID` property. The FilteredTextBoxExtender control has a property called `FilterType` that has the possible values of `Custom`, `LowercaseLetters`, `Numbers`, and `UppercaseLetters`.

This example uses a `FilterType` value of `Numbers`, meaning that only numbers can be entered into the text box. If the end user tries to enter any other type of information, then nothing happens — it will seem to the end user as if the key doesn't even function.

Another property the FilteredTextBoxExtender control exposes is the `FilterMode` and the `InvalidChars` properties. An example of using these two properties is presented here:

```
<cc1:FilteredTextBoxExtender ID="FilteredTextBoxExtender1" runat="server"
 TargetControlID="TextBox1" InvalidChars="*" FilterMode="InvalidChars">
</cc1:FilteredTextBoxExtender>
```

The default value of the `FilterMode` property is `ValidChars`. When set to `ValidChars`, the control works from the `FilterType` property and only allows what this property defines. When set to `InvalidChars`, you then use the `InvalidChars` property and put the characters here (multiple characters all go together with no space or item between them).

HoverMenuExtender

The HoverMenuExtender control allows you to make a hidden control appear on the screen when the end user hovers on another control. This means that you can build either elaborate ToolTips or provide extra functionality when an end user hovers somewhere in your application.

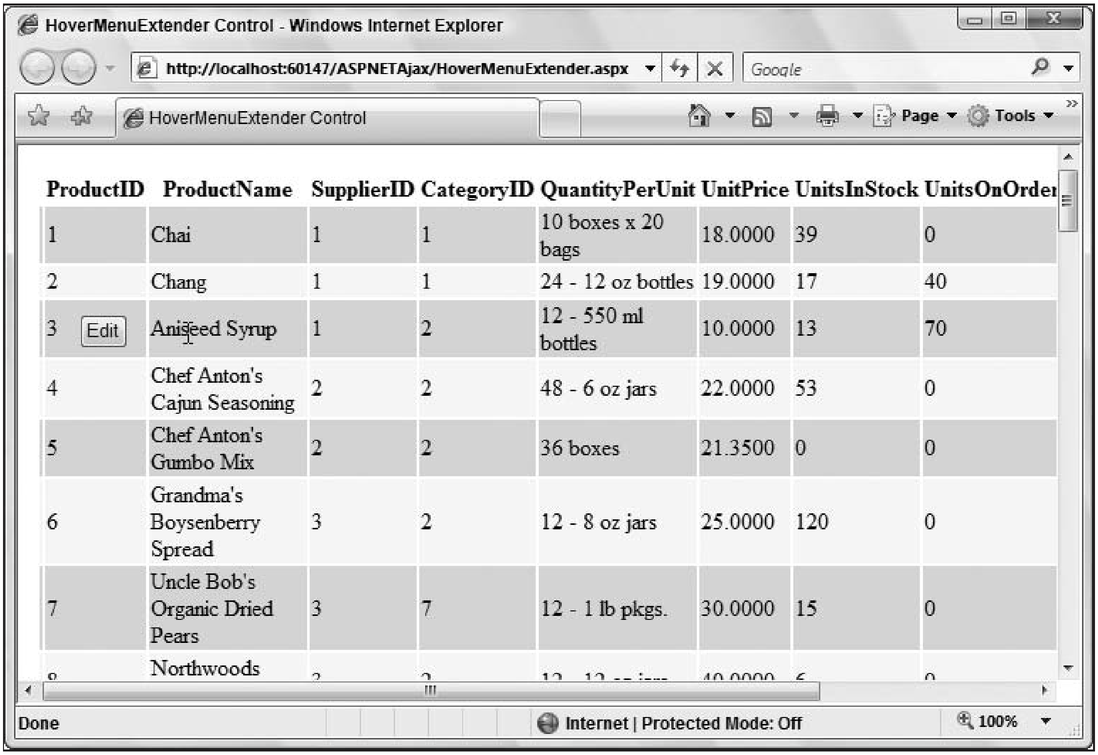
One example is to change a ListView control so that when the end user hovers over a product name, the Edit button for that row of data appears on the screen. The code for the `<ItemTemplate>` in the ListView control is partially shown in Listing 20-17.

Listing 20-17: Adding a hover button to the ListView control's ItemTemplate

```
<ItemTemplate>
   <tr style="background-color:#DCDCDC;color: #000000;">
      <td>
         <cc1:HoverMenuExtender ID="HoverMenuExtender1" runat="server"
          TargetControlID="ProductNameLabel" PopupControlID="Panel1"
          PopDelay="25" OffsetX="-50">
         </cc1:HoverMenuExtender>
         <asp:Panel ID="Panel1" runat="server" Height="50px" Width="125px">
            <asp:Button ID="EditButton" runat="server"
             CommandName="Edit" Text="Edit" />
         </asp:Panel>
      </td>
      <td>
         <asp:Label ID="ProductIDLabel" runat="server"
          Text='<%# Eval("ProductID") %>' />
      </td>
      <td>
         <asp:Label ID="ProductNameLabel" runat="server"
          Text='<%# Eval("ProductName") %>' />
      </td>

      <!-- Code removed for clarity -->

   </tr>
</ItemTemplate>
```

Here, a HoverMenuExtender control is attached to the Label control with the `ID` of `ProductNameLabel`, which appears in each row of the ListView control. This is done using the `TargetControlID` property, while the `PopupControlID` property is used to assign the control that appears dynamically when a user hovers the mouse over the targeted control.

The HoverMenuExtender control exposes several properties that control the style and behaviors of the popup. First off, the `PopDelay` property is used in this example and provides a means to delay the popup from occurring (in milliseconds). The `OffsetX` and `OffsetY` properties specify the location of the popup based upon the targeted control. In this case, the offset is set to `-50` (pixels). The results of the operation are shown in Figure 20-22.


HoverMenuExtender Control - Windows Internet Explorer
http://localhost:60147/ASPNETAjax/HoverMenuExtender.aspx
Google
HoverMenuExtender Control
Page
Tools
Edit
Done
Internet | Protected Mode: Off
100%

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder
1	Chai	1	1	10 boxes x 20 bags	18.0000	39	0
2	Chang	1	1	24 - 12 oz bottles	19.0000	17	40
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10.0000	13	70
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22.0000	53	0
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.3500	0	0
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25.0000	120	0
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30.0000	15	0
	Northwoods						

Figure 20-22

ListSearchExtender

The ListSearchExtender control extends either a ListBox or a DropDownList control, though not always with the best results in browsers such as Opera and Safari. This extender allows you to provide search capabilities through large collections that are located in either of these controls. This alleviates the need for the end users to search through the collection to find the item they are looking for.

When utilized, the extender adds a search text that shows the characters the end user types for their search area above the control. Listing 20-18 shows the use of this extender.

Listing 20-18: Extending a ListBox control with the ListSearchExtender control

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>ListSearchExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:ListSearchExtender ID="ListSearchExtender1" runat="server"
         TargetControlID="ListBox1">
        </cc1:ListSearchExtender>
        <asp:ListBox ID="ListBox1" runat="server" Width="150">
            <asp:ListItem>Aardvark</asp:ListItem>
            <asp:ListItem>Bee</asp:ListItem>
            <asp:ListItem>Camel</asp:ListItem>
            <asp:ListItem>Dog</asp:ListItem>
            <asp:ListItem>Elephant</asp:ListItem>
        </asp:ListBox>
    </div>
    </form>
</body>
</html>
```

In this case, the only property used in the ListSearchExtender control is the `TargetControlID` property to associate which control it extends. Running this page produces the results shown in Figure 20-23.

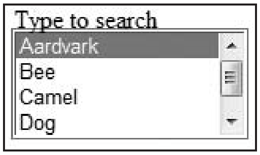


Figure 20-23

Then, as an end user, when you start typing, you will see what you are typing in the text above the control (as shown in Figure 20-24).

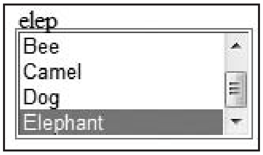


Figure 20-24

You can customize the text that appears at the top of the control with the `PromptCssClass`, `PromptPosition`, and `PromptText` properties. By default, the `PromptPosition` is set to `Top` (the other possible value is `Bottom`) and the `PromptText` value is `Type to search`.

MaskedEditExtender and MaskedEditValidator

The MaskedEditExtender control is similar to the FilteredTextBoxExtender control in that it restricts the end user from entering specific text within a TextBox control. This control takes the process one step further in that it provides the end user a template within the text box for them to follow. If the end users do not follow the template, they will be unable to proceed and might get a validation warning from the control using the MaskedEditValidator control.

Listing 20-19 provides an example of using both of these controls.

Listing 20-19: Using both the MaskedEditExtender and the MaskedEditValidator controls

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit, Version=3.5.11119.20050,
    Culture=neutral, PublicKeyToken=28f01b0e84b6d53e"
    Namespace="AjaxControlToolkit" TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>MaskedEditExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:MaskedEditExtender ID="MaskedEditExtender1" runat="server"
         TargetControlID="TextBox1" MaskType="Number" Mask="999">
        </cc1:MaskedEditExtender>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <cc1:MaskedEditValidator ID="MaskedEditValidator1" runat="server"
         ControlExtender="MaskedEditExtender1" ControlToValidate="TextBox1"
         IsValidEmpty="False" EmptyValueMessage="A three digit number is required!"
         Display="Dynamic"></cc1:MaskedEditValidator>
    </div>
    </form>
</body>
</html>
```

In this case, the MaskedEditExtender control uses the `TargetControlID` to associate itself with the TextBox1 control. The `MaskType` property supplies the type of mask or filter to place on the text box. The possible values include:

- ❑ `None`–means that no validation will be performed.
- ❑ `Date`–means date validation will occur.

- `DateTime`–means date and time validation will occur.
- `Number`–means a number validation will occur.
- `Time`–means a time validation will occur

Listing 20-19 uses `Number` and then specifies the mask or template the numbers need to take. This is done through the use of the `Mask` property. In this case, the `Mask` property is set to `999`. This means that all numbers can be only three digits in length.

Using `999` as a value to the `Mask` property means that when end user enters a value in the text box, he will be presented with three underscores inside the text box. This is the template for entering items. This is shown in Figure 20-25.

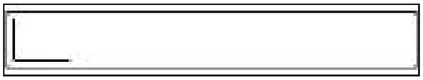


Figure 20-25

If the `Mask` property is changed to `99,999.99` as follows:

```
<cc1:MaskedEditExtender ID="MaskedEditExtender1" runat="server"
 TargetControlID="TextBox1" MaskType="Number" Mask="99,999.99">
</cc1:MaskedEditExtender>
```

. . . the textbox template appears as illustrated in Figure 20-26.

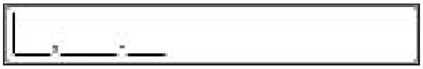


Figure 20-26

From Figure 20.26, you can see that the comma and the period are present in the template. As the end users type, they do not need to retype these values. The cursor will simply move to the next section of numbers required.

As you can see from the `Mask` property value, numbers are represented by the number `9`. When working with other `MaskType` values, you also need to be aware of the other mask characters. These are provided here in the following list:

- `9` – Only a numeric character
- `L` – Only a letter
- `$` – Only a letter or a space
- `C` – Only a custom character (case sensitive)
- `A` – Only a letter or a custom character
- `N` – Only a numeric or custom character
- `?` – Any character

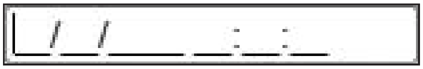
In addition to the character specifications, the delimiters used in the template are detailed in the following list:

- / is a date separator
- : is a time separator
- . is a decimal separator
- , is a thousand separator
- \ is the escape character
- { is the initial delimiter for repetition of masks
- } is the final delimiter for repetition of masks

Using some of these items, you can easily change MaskedEditExtender to deal with a `DateTime` value:

```
<cc1:MaskedEditExtender ID="MaskedEditExtender1" runat="server"
 TargetControlID="TextBox1" MaskType="DateTime" Mask="99/99/9999 99:99:99">
</cc1:MaskedEditExtender>
```

The template created in the text box for this is shown in Figure 20-27.



__/__/____ __:__:__

Figure 20-27

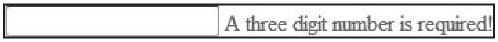
The MaskedEditExtender control has a ton of properties that are exposed to control and manipulate the behavior and style of the text box. The MaskedEditExtender control can work in conjunction with the MaskedEditValidator control, which provides validation against the text box controls.

In the earlier example, the validation was accomplished through an instance of the MaskedEditValidator control.

```
<cc1:MaskedEditValidator ID="MaskedEditValidator1" runat="server"
 ControlExtender="MaskedEditExtender1" ControlToValidate="TextBox1"
 IsValidEmpty="False" EmptyValueMessage="A three digit number is required!"
 Display="Dynamic"></cc1:MaskedEditValidator>
```

This control uses the ControlExtender property to associate itself with the MaskedEditExtender control and uses the `ControlToValidate` property to watch a specific control on the form. By default, the `IsValidEmpty` property is set to `True`, but changing it to `False` means that the end user will be required to enter some value in the text box in order to pass validation and not get the error message that is presented in the `EmptyValueMessage` property.

Triggering the MaskedEditValidator control gives you something like the message shown in Figure 20-28. It is important to remember that you can style the control in many ways to produce the validation message appearance that you are looking for.



A three digit number is required!

Figure 20-28

MutuallyExclusiveCheckBoxExtender

Often you want to offer a list of check boxes that behave as if they are radio buttons. That is, when you have a collection of check boxes, you will only want the end user to make a single selection from the provided list of items. However, unlike a radio button, you also want to enable the end user to deselect an item or to make no selection whatsoever.

Using the MutuallyExclusiveCheckBoxExtender control, you can perform such an action. Listing 20-20 shows you how to accomplish this task.

Listing 20-20: Using the MutuallyExclusiveCheckBoxExtender control with check boxes

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>MutuallyExclusiveCheckBoxExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:MutuallyExclusiveCheckBoxExtender
         ID="MutuallyExclusiveCheckBoxExtender1" runat="server"
         TargetControlID="CheckBox1" Key="MyCheckboxes" />
        <asp:CheckBox ID="CheckBox1" runat="server" Text="Blue" /><br />
        <cc1:MutuallyExclusiveCheckBoxExtender
         ID="MutuallyExclusiveCheckBoxExtender2" runat="server"
         TargetControlID="CheckBox2" Key="MyCheckboxes" />
        <asp:CheckBox ID="CheckBox2" runat="server" Text="Brown" /><br />
        <cc1:MutuallyExclusiveCheckBoxExtender
         ID="MutuallyExclusiveCheckBoxExtender3" runat="server"
         TargetControlID="CheckBox3" Key="MyCheckboxes" />
        <asp:CheckBox ID="CheckBox3" runat="server" Text="Green" /><br />
        <cc1:MutuallyExclusiveCheckBoxExtender
         ID="MutuallyExclusiveCheckBoxExtender4" runat="server"
         TargetControlID="CheckBox4" Key="MyCheckboxes" />
        <asp:CheckBox ID="CheckBox4" runat="server" Text="Orange" /><br />
    </div>
    </form>
</body>
</html>
```

It is impossible to associate a MutuallyExclusiveCheckBoxExtender control with a CheckBoxList control, therefore, each of the check boxes needs to be laid out with CheckBox controls as the previous code demonstrates. You need to have one MutuallyExclusiveCheckBoxExtender control for each CheckBox control on the page.

You form a group of CheckBox controls by using the `Key` property. All the check boxes that you want in one group need to have the same `Key` value. In the example is Listing 20-20, all the check boxes share a `Key` value of `MyCheckboxes`.

Running this page, results in a list of four check boxes. When you select one of the check boxes, a check mark appears. Then, when you select another check box, first checkbox you selected gets deselected. The best part is that you can even deselect what you have selected in the group, thereby selecting nothing in the check box group.

NumericUpDownExtender

The NumericUpDownExtender control allows you to put some up/down indicators next to a TextBox control that enable the end user to more easily control a selection.

A simple example of this is illustrated here in Listing 20-21.

Listing 20-21: Using the NumericUpDownExtender control

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>NumericUpDownExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:NumericUpDownExtender ID="NumericUpDownExtender1" runat="server"
         TargetControlID="TextBox1" Width="150" Maximum="10" Minimum="1">
        </cc1:NumericUpDownExtender>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </div>
    </form>
</body>
</html>
```

The NumericUpDownExtender control here extends the TextBox control on the page. When using the NumericUpDownExtender control, you have to specify the width of the control with the `Width` property. Otherwise, you will see only the up and down arrow keys and not the text box area. In this case, the `Width` property is set to `150` (pixels). The `Maximum` and `Minimum` properties provide the range used by the up and down indicators.

With a `Maximum` value setting of `10` and a `Minimum` value of `1`, the only range in the control will be 1 through 10. Running this page produces the results shown in Figure 20-29.

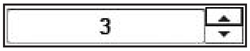


Figure 20-29

In addition to numbers as is shown with Listing 20-21, you can also use text as is illustrated here in Listing 20-22.

Listing 20-22: Using characters instead of numbers with NumericUpDownExtender

```
<cc1:NumericUpDownExtender ID="NumericUpDownExtender1" runat="server"
 TargetControlID="TextBox1" Width="150"
 RefValues="Blue;Brown;Green;Orange;Black;White">
</cc1:NumericUpDownExtender>
```

In this case, the words are defined within the `RefValues` property (all separated with a semicolon). This gives you the results presented in Figure 20-30.

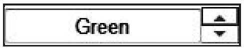


Figure 20-30

PagingBulletedListExtender

The PagingBulletedListExtender control allows you to take long bulleted lists and easily apply alphabetic paging to the list. For an example of this, Listing 20-23 will work off of the Customers table within the Northwind database.

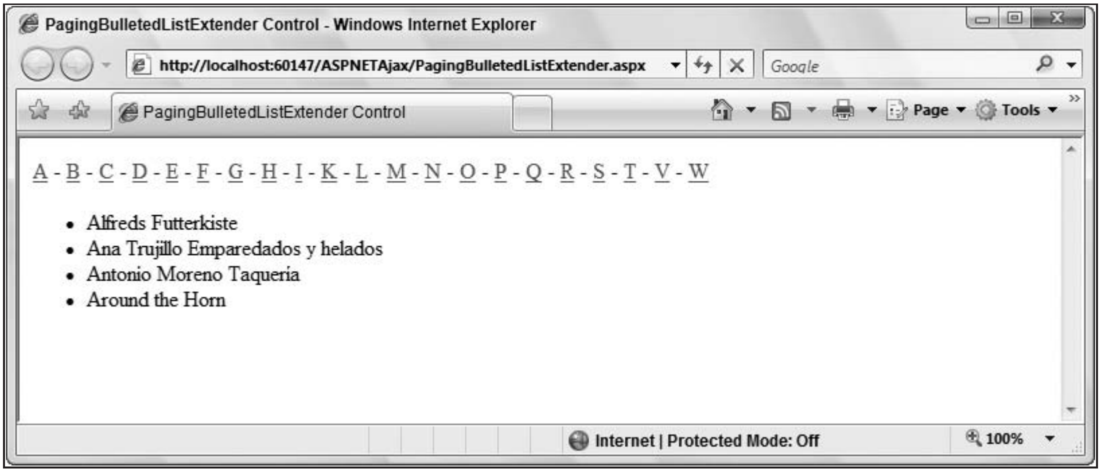
Listing 20-23: Paging a bulleted list from the Northwind database

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>PagingBulletedListExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:PagingBulletedListExtender ID="PagingBulletedListExtender1"
         runat="server" TargetControlID="BulletedList1">
        </cc1:PagingBulletedListExtender>
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="Data Source=.\SQLEXPRESS;
```

Continued

```
                AttachDbFilename=|DataDirectory|\NORTHWND.MDF;
                Integrated Security=True;User Instance=True"
             ProviderName="System.Data.SqlClient"
             SelectCommand="SELECT [CompanyName] FROM [Customers]">
         </asp:SqlDataSource>
         <asp:BulletedList ID="BulletedList1" runat="server"
             DataSourceID="SqlDataSource1" DataTextField="CompanyName"
             DataValueField="CompanyName">
         </asp:BulletedList>
     </div>
     </form>
</body>
</html>
```

This code pulls all the `CompanyName` values from the Customers table of the Northwind database and binds those values to the BulletList control on the page. Running this page gives you the results illustrated in Figure 20-31.


PagingBulletedListExtender Control - Windows Internet Explorer
http://localhost:60147/ASPNETAjax/PagingBulletedListExtender.aspx
Google
PagingBulletedListExtender Control
Page
Tools
Internet | Protected Mode: Off
100%

A - B - C - D - E - F - G - H - I - K - L - M - N - O - P - Q - R - S - T - V - W

- Alfreds Futterkiste
- Ana Trujillo Emparedados y helados
- Antonio Moreno Taqueria
- Around the Horn

Figure 20-31

From this figure, you can see that the paging is organized alphabetically on the client side. Only the letters for which there are values appear in the linked list of letters. Clicking any of the letters gives you the items from the bulleted list that start with that character.

PopupControlExtender

The PopupControlExtender control allows you to create a popup for any control on your page. For instance, you can completely mimic the CalendarExtender control that was presented earlier by creating a popup containing a Calendar control off a TextBox control. Listing 20-24 mimics this behavior.

Listing 20-24: Creating a CalendarExtender control with PopupControlExtender

VB

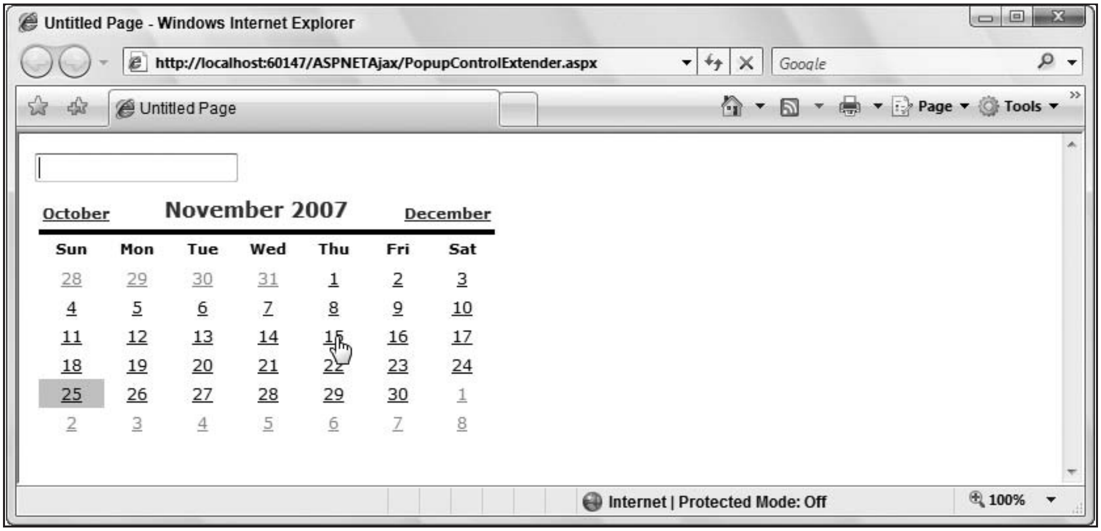
```
<%@ Page Language="VB" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<script runat="server">
    Protected Sub Calendar1_SelectionChanged(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        PopupControlExtender1.Commit(Calendar1.SelectedDate.ToShortDateString())
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>PopupControlExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:PopupControlExtender ID="PopupControlExtender1" runat="server"
         TargetControlID="TextBox1" PopupControlID="UpdatePanel1" OffsetY="25">
        </cc1:PopupControlExtender>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <asp:Calendar ID="Calendar1" runat="server" BackColor="White"
                 BorderColor="White" BorderWidth="1px" Font-Names="Verdana"
                 Font-Size="9pt" ForeColor="Black" Height="190px"
                 NextPrevFormat="FullMonth" Width="350px"
                 OnSelectionChanged="Calendar1_SelectionChanged">
                    <SelectedDayStyle BackColor="#333399" ForeColor="White" />
                    <TodayDayStyle BackColor="#CCCCCC" />
                    <OtherMonthDayStyle ForeColor="#999999" />
                    <NextPrevStyle Font-Bold="True" Font-Size="8pt"
                     ForeColor="#333333" VerticalAlign="Bottom" />
                    <DayHeaderStyle Font-Bold="True" Font-Size="8pt" />
                    <TitleStyle BackColor="White" BorderColor="Black"
                     BorderWidth="4px" Font-Bold="True" Font-Size="12pt"
                     ForeColor="#333399" />
                </asp:Calendar>
            </ContentTemplate>
        </asp:UpdatePanel>
    </div>
    </form>
</body>
</html>
```

Continued

C#

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<script runat="server">
    protected void Calendar1_SelectionChanged(object sender, EventArgs e)
    {
        PopupControlExtender1.Commit(Calendar1.SelectedDate.ToShortDateString());
    }
</script>
```

When running this page, you get a single text box on the page. Click within the text box and a pop-up calendar appears so you can select a date that will be populated back into the text box (as illustrated in Figure 20-32).


Untitled Page - Windows Internet Explorer
http://localhost:60147/ASPNETAjax/PopupControlExtender.aspx
Google
Untitled Page
Page
Tools
Internet | Protected Mode: Off
100%

October			November 2007			December
Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	1
2	3	4	5	6	7	8

Figure 20-32

You will want to place your popup control within an ASP.NET AJAX UpdatePanel control and to pass the value from the popup control back to the target control (the TextBox1 control), so you use the `Commit()` method:

```
PopupControlExtender1.Commit(Calendar1.SelectedDate.ToShortDateString())
```

ResizableControlExtender

The ResizableControlExtender control allows you to take a Panel control and give end users the ability to grab a handle and change the size of the element (smaller or bigger). Anything you put inside the `Panel` control will then change in size depending on how the end user extends the item. For this to work, you have to create a handle for the end user to use. Listing 20-25 shows you how to use ResizableControlExtender with an image.

Listing 20-25: Using the ResizableControlExtender control with an image

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>ResizableControlExtender Control</title>
    <style type="text/css">
        .handle
        {
         width:10px;
         height:10px;
         background-color:Black;
        }
        .resizable
        {
            border-style:solid;
            border-width:2px;
            border-color:Black;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:ResizableControlExtender ID="ResizableControlExtender1" runat="server"
         TargetControlID="Panel1" HandleCssClass="handle"
         ResizableCssClass="resizable">
        </cc1:ResizableControlExtender>
        <asp:Panel ID="Panel1" runat="server" Width="300" Height="225">
            <asp:Image ID="Image1" runat="server" ImageUrl="Images/Garden.jpg"
             style="width:100%; height:100%"/>
        </asp:Panel>
    </div>
    </form>
</body>
</html>
```

In this example, the ResizableControlExtender control depends upon CSS to create the handle for the end user to grab to resize the `Panel` control. The `TargetControlID` property points to the control to be resized.

There are two CSS references in the ResizableControlExtender control. One deals with the control as it sits on the screen with no end user interaction. This is really to show the end user that there is an ability to resize the element. This is done through the `HandleCssClass` property. The value of this property points to the CSS class `handle` contained within the same file. The second CSS reference deals with the control as it is clicked and held (when the end user does not let up with the mouse click performed). This one is done with the `ResizableCssClass` property. The value of this property points to the CSS class `resizable`.

When compiled and run, the code should generate the same page presented in Figure 20-33.

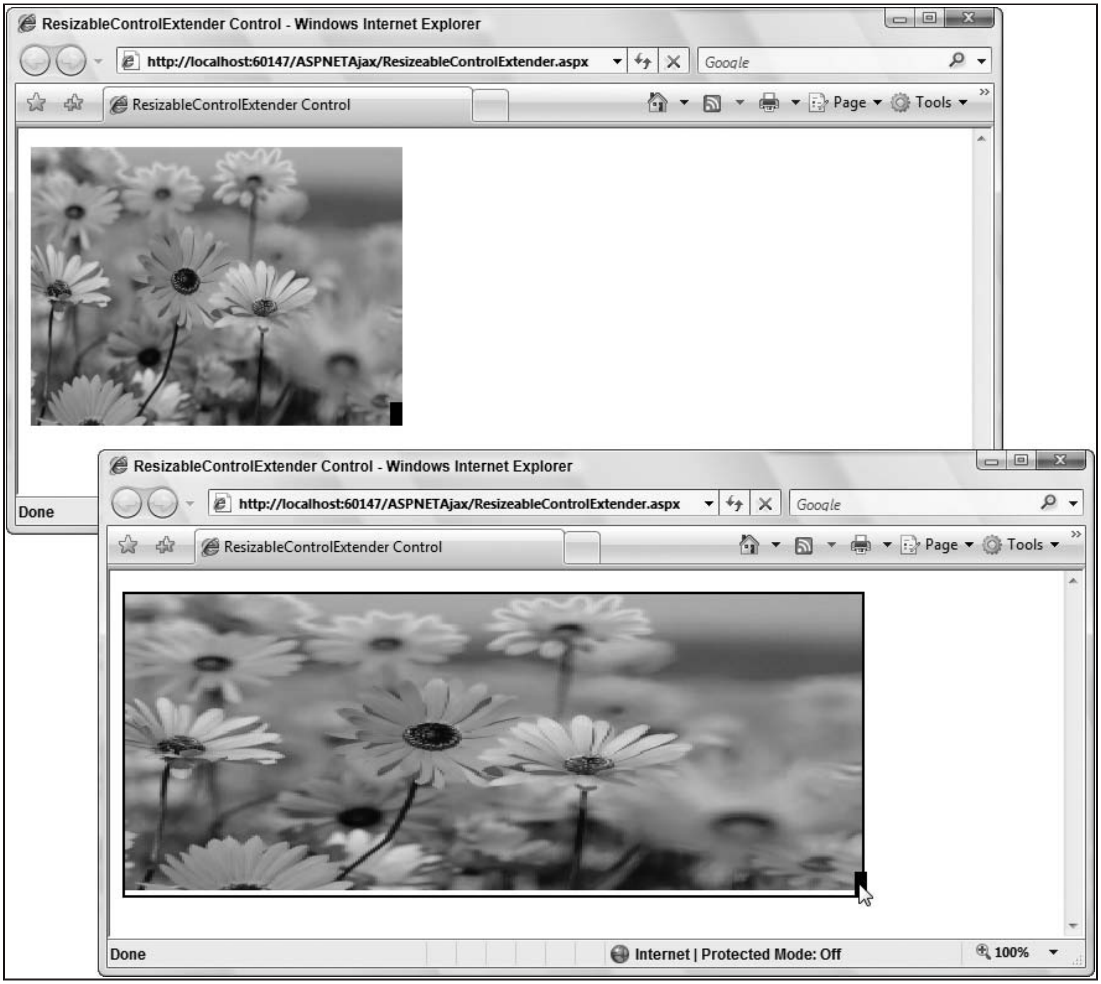

ResizableControlExtender Control - Windows Internet Explorer
http://localhost:60147/ASPNETAjax/ResizeableControlExtender.aspx
Google
ResizableControlExtender Control
Page
Tools
Done
ResizableControlExtender Control - Windows Internet Explorer
http://localhost:60147/ASPNETAjax/ResizeableControlExtender.aspx
Google
ResizableControlExtender Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

Figure 20-33

You can see in the top screenshot how the image looks when there is no end user interaction. In this case, there is a black square (as defined by the CSS) in the lower-right corner of the image. The screenshot on the bottom shows what happens when the end user grabs the handle and starts changing the shape of the image.

RoundedCornersExtender

The RoundedCornersExtender control allows you to put rounded corners on the elements on your page. As with the ResizableControlExtender control, you put the element you are interested in working with inside of a Panel control. Listing 20-26 shows this done with a Login server control.

Listing 20-26: Rounding the corners of the Panel control containing a Login server control

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>RoundedCornersExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:RoundedCornersExtender ID="RoundedCornersExtender1" runat="server"
         TargetControlID="Panel1">
        </cc1:RoundedCornersExtender>
        <asp:Panel ID="Panel1" runat="server" Width="250px"
         HorizontalAlign="Center" BackColor="Orange">
            <asp:Login ID="Login1" runat="server">
            </asp:Login>
        </asp:Panel>
    </div>
    </form>
</body>
</html>
```

Here, the RoundedCornersExtender control simply points to the Panel control with the `TargetControlID` property. This Panel control has a background color of orange to show that the corners are indeed rounded. The result of this bit of code is illustrated in Figure 20-34.

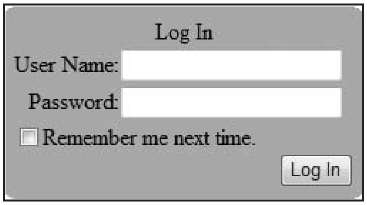


Figure 20-34

You can control the degree of the rounded corners using the `Radius` property of the RoundedCornersExtender control. By default, this property is set to a value of `5`. You can even go as far as choosing the corners that you want to round using the `Corners` property. The possible values of the `Corners` property include `All`, `Bottom`, `BottomLeft`, `BottomRight`, `Left`, `None`, `Right`, `Top`, `TopLeft`, and `TopRight`.

SliderExtender

The SliderExtender control actually extends a TextBox control to make it look like nothing it normally does. This ASP.NET AJAX control gives you the ability to create a true slider control that allows the end user to select a range of numbers using a mouse instead typing in the number. Listing 20-27 shows a simple example of using the slider.

Listing 20-27: Using the SliderExtender control

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>SliderExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:SliderExtender ID="SliderExtender1" runat="server"
         TargetControlID="TextBox1">
        </cc1:SliderExtender>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </div>
    </form>
</body>
</html>
```

This little bit of code to tie a SliderExtender control to a typical TextBox control is simple and produces the result presented in Figure 20-35.

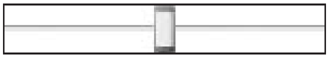


Figure 20-35

This is fine, but it is hard for the end users to tell what number they are selecting. Therefore, you might find it better to give a signifier to the end user. Adding a Label control to the page (called `Label1`) and changing the SliderExtender control to include a `BoundControlID` property gives you this result. The code for this change is presented here:

```
<cc1:SliderExtender ID="SliderExtender1" runat="server" TargetControlID="TextBox1"
 BoundControlID="Label1">
</cc1:SliderExtender>
```

This small change produces see the result (with the appropriate Label control on the page) shown in Figure 20-36.

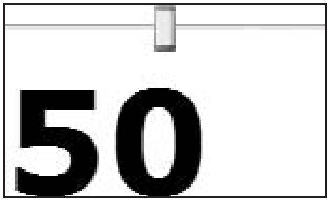


Figure 20-36

Now when the end users slide the handle on the slider, they see the number that they are working with quite easily. Some of the following properties are available to the SliderExtender control.

- `Decimal` – Allows you to specify the number of decimals the result should take. The more decimals you have, the more unlikely the end user will be able to pick an exact number.
- `HandleCssClass` – The CSS class that you are using the design the handle.
- `HandleImageUrl` – The image file you are using to represent the handle.
- `Length` – The length of the slider in pixels. The default value is `150`.
- `Maximum` – The maximum number represented in the slider. The default value is `100`.
- `Minimum` – The minimum number represented in the slider. The default value is `0`.
- `Orientation` — The orientation of the slider. The possible values include `Horizontal` and `Vertical`. The default value is `Horizontal`.
- `RailCssClass` — The CSS class that you are using to design the rail of the slider.
- `ToolTipText` — The ToolTip when the end user hovers over the slider. Using `0` within the text provides the means show the end user the position the slider is currently in.

SlideShowExtender

The SlideShowExtender control allows you to put an image slideshow in the browser. The slideshow controls allow the end user to move to the next or previous images as well as simply play the images as a slideshow with a defined wait between each image. Listing 20-28 shows an example of creating a slideshow.

Listing 20-28: Creating a slideshow with three images

.ASPX

```
<%@ Page Language="VB" AutoEventWireup="true" CodeFile="SlideShowExtender.aspx.vb"
    Inherits="SlideShowExtender" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

Continued

```
        <title>SlideShowExtender Control</title>
    </head>
    <body>
        <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <asp:Panel ID="Panel1" runat="server" Width="300px"
             HorizontalAlign="Center">
                <cc1:SlideShowExtender ID="SlideShowExtender1" runat="server"
                    ImageTitleLabelID="LabelTitle" TargetControlID="Image1"
                    UseContextKey="True" NextButtonID="ButtonNext"
                    PlayButtonID="ButtonPlay"
                    PreviousButtonID="ButtonPrevious"
                    SlideShowServiceMethod="GetSlides"
                    ImageDescriptionLabelID="LabelDescription">
                </cc1:SlideShowExtender>
                <asp:Label ID="LabelTitle" runat="server" Text="Label"
                 Font-Bold="True"></asp:Label><br /><br />
                <asp:Image ID="Image1" runat="server"
                 ImageUrl="Images/Garden.jpg" /><br />
                <asp:Label ID="LabelDescription" runat="server"
                 Text="Label"></asp:Label><br /><br />
                <asp:Button ID="ButtonPrevious" runat="server" Text="Previous" />
                <asp:Button ID="ButtonNext" runat="server" Text="Next" />
                <asp:Button ID="ButtonPlay" runat="server" />
            </asp:Panel>
        </div>
        </form>
    </body>
    </html>
```

The SlideShowExtender control has a lot of available properties available. You can specify the location where you are defining the image title and description using the `ImageTitleLabelID` and the `ImageDescriptionLabelID` properties. In addition to that, this page contains three Button controls. One to act as the Previous button, another for the Next button, and the final one as the Play button. However, it is important to note that when the Play button is clicked (to start the slideshow), it turns into the Stop button.

The `SlideShowServiceMethod` property is important because it points to the server-side method that returns the images that are part of the slide show. In this case, it is referring to a method called GetSlides, which is represented here in Listing 20-29.

Listing 20-29: The GetSlides method implementation

VB

```
Partial Class SlideShowExtender
    Inherits System.Web.UI.Page

    <System.Web.Services.WebMethodAttribute()> _
    <System.Web.Script.Services.ScriptMethodAttribute()> _
    Public Shared Function GetSlides(ByVal contextKey As System.String) _
      As AjaxControlToolkit.Slide()

               Return New AjaxControlToolkit.Slide() { _
                  New AjaxControlToolkit.Slide("Images/Creek.jpg",
                     "The Creek", "This is a picture of a creek."),
                  New AjaxControlToolkit.Slide("Images/Dock.jpg",
                     "The Dock", "This is a picture of a Dock."),
                  New AjaxControlToolkit.Slide("Images/Garden.jpg",
                     "The Garden", "This is a picture of a Garden.") }
    End Function
End Class
```

C#

```
public partial class SlideShowExtender : System.Web.UI.Page
{
    [System.Web.Services.WebMethodAttribute(),
     System.Web.Script.Services.ScriptMethodAttribute()]
    public static AjaxControlToolkit.Slide[] GetSlides(string contextKey)
    {
        return new AjaxControlToolkit.Slide[] {
            new AjaxControlToolkit.Slide("Images/Creek.jpg",
              "The Creek", "This is a picture of a creek."),
            new AjaxControlToolkit.Slide("Images/Dock.jpg",
              "The Dock", "This is a picture of a Dock."),
            new AjaxControlToolkit.Slide("Images/Garden.jpg",
              "The Garden", "This is a picture of a Garden.") };
    }
}
```

With the code-behind in place, the SlideShowExtender has a server-side method to call for the photos. This method, called `GetSlides()`, returns an array of `Slide` objects which require the location of the object (the path), the title, and the description. When running this page, you get something similar to the following results shown in Figure 20-37.

Pressing the Play button on the page will rotates the images until they are done. They will not repeat in a loop unless you have the SlideShowExtender control's `Loop` property set to `True`. (It is set to `False` by default.)

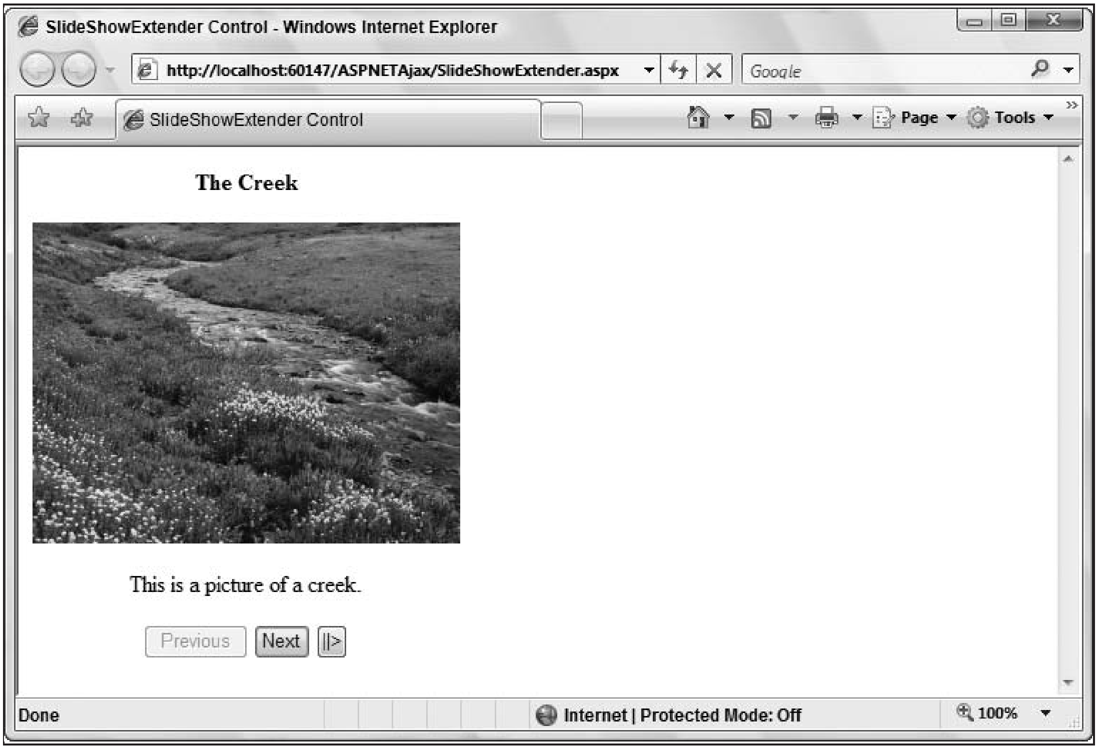
The other important property to pay attention to is the `PlayInterval` property. The value of this property is an integer that represents the number of milliseconds that the browser will take to change to the next photo in the series of images. By default, this is set to `3000` milliseconds.

TextBoxWatermarkExtender

The TextBoxWatermarkExtender control allows you to put instructions within controls for the end users, which gives them a better understanding of what to use the control for. This can be text or even images (when using CSS). Listing 20-30 shows an example of using this control with a TextBox server control.

Listing 20-30: Using the TextBoxWatermarkExtender control with a TextBox control

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>TextBoxWatermarkExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:TextBoxWatermarkExtender ID="TextBoxWatermarkExtender1" runat="server"
         WatermarkText="Enter in something here!" TargetControlID="TextBox1">
        </cc1:TextBoxWatermarkExtender>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </div>
    </form>
</body>
</html>
```

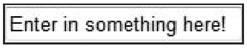

SlideShowExtender Control - Windows Internet Explorer
http://localhost:60147/ASPNETAjax/SlideShowExtender.aspx
Google
SlideShowExtender Control
Page
Tools
Previous
Next
|>
Done
Internet | Protected Mode: Off
100%

The Creek

This is a picture of a creek.

Figure 20-37

In this case, the TextBoxWatermarkExtender control is associated with a simple TextBox control and uses the `WatermarkText` property to provide the text that will appear inside the actual TextBox control. Figure 20-38 shows the results of the code from this listing.



Enter in something here!

Figure 20-38

The text in the image from Figure 20-38 is straight text with no style inside of the TextBox control. When the end user clicks inside of the TextBox control, the text will disappear and the cursor will be properly placed at the beginning of the text box.

To apply some style to the content that you use as a watermark, you can use the `WatermarkCssClass` property. You can change the code to include a bit of style as shown in Listing 20-31.

Listing 20-31: Applying style to the watermark

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>TextBoxWatermarkExtender Control</title>
    <style type="text/css">
        .watermark
        {
         width:150px;
         font:Verdana;
         font-style:italic;
         color:GrayText;
        }

    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:TextBoxWatermarkExtender ID="TextBoxWatermarkExtender1" runat="server"
         WatermarkText="Enter in something here!" TargetControlID="TextBox1"
         WatermarkCssClass="watermark">
        </cc1:TextBoxWatermarkExtender>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </div>
    </form>
</body>
</html>
```

This time, the WatermarkCssClass property is used and points to the inline CSS class, `watermark`, which is on the page. Running this page, you will see the style applied as shown in Figure 20-39.

Enter in something here!

Figure 20-39

ToggleButtonExtender

The ToggleButtonExtender control works with CheckBox controls and allows you to use an image of your own instead of the standard check box images that the CheckBox controls typically use. Using the ToggleButtonExtender control, you are able to specify images for checked, unchecked, and disabled statuses. Listing 20-32 shows an example of using this control.

Listing 20-32: Using the ToggleButtonExtender control

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>ToggleButtonExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:MutuallyExclusiveCheckBoxExtender
         ID="MutuallyExclusiveCheckBoxExtender1" runat="server" Key="MyCheckBoxes"
         TargetControlID="CheckBox1">
        </cc1:MutuallyExclusiveCheckBoxExtender>
        <cc1:MutuallyExclusiveCheckBoxExtender
         ID="MutuallyExclusiveCheckBoxExtender2" runat="server" Key="MyCheckBoxes"
         TargetControlID="CheckBox2">
        </cc1:MutuallyExclusiveCheckBoxExtender>
        <cc1:ToggleButtonExtender ID="ToggleButtonExtender1" runat="server"
         TargetControlID="CheckBox1" UncheckedImageUrl="Images/Unchecked.gif"
         CheckedImageUrl="Images/Checked.gif" CheckedImageAlternateText="Checked"
         UncheckedImageAlternateText="Not Checked" ImageWidth="25"
         ImageHeight="25">
        </cc1:ToggleButtonExtender>
        <asp:CheckBox ID="CheckBox1" runat="server" Text="&nbsp;Option One" />
        <cc1:ToggleButtonExtender ID="ToggleButtonExtender2" runat="server"
         TargetControlID="CheckBox2" UncheckedImageUrl="Images/Unchecked.gif"
         CheckedImageUrl="Images/Checked.gif" CheckedImageAlternateText="Checked"
         UncheckedImageAlternateText="Not Checked" ImageWidth="25"
         ImageHeight="25">
        </cc1:ToggleButtonExtender>
        <asp:CheckBox ID="CheckBox2" runat="server" Text="&nbsp;Option Two" />
    </div>
    </form>
</body>
</html>
```

This page has two CheckBox controls. Each check box has an associated ToggleButtonExtender control along with a MutuallyExclusiveCheckBoxExtender control to tie the two check boxes together. The ToggleButtonExtender control uses the `CheckedImageUrl` and the `UncheckedImageUrl` properties to specify the appropriate images to use. Then, if images are disabled by the end user's browser instance, the text that is provided in the `CheckedImageAlternateText` and `UncheckedImageAlternateText` properties is used instead. You will also have to specify values for the `ImageWidth` and `ImageHeight` properties for the page to run.

Running this page, you get results similar to those presented in Figure 20-40.

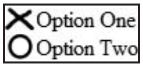


Figure 20-40

UpdatePanelAnimationExtender

The UpdatePanelAnimationExtender control allows you to apply an animation to a Panel control for two specific events. The first is the `OnUpdating` event and the second is the `OnUpdated` event. You can then use the animation framework provided by ASP.NET AJAX to change the page's style based on these two events. Listing 20-33 shows an example of using the `OnUpdated` event when the end user clicks a specific date within a `Calendar` control contained within the UpdatePanel control on the page.

Listing 20-33: Using animations on the OnUpdated event

VB

```
<%@ Page Language="VB" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<script runat="server">
    Protected  Sub Calendar1_SelectionChanged(ByVal sender As Object, _
      ByVal e As EventArgs)
        Label1.Text = "The date selected is " &
           Calendar1.SelectedDate.ToLongDateString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>UpdatePanelAnimationExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:UpdatePanelAnimationExtender ID="UpdatePanelAnimationExtender1"
         runat="server" TargetControlID="UpdatePanel1">
            <Animations>
```

Continued

```
                <OnUpdated>
                    <Sequence>
                        <Color PropertyKey="background" StartValue="#999966"
                         EndValue="#FFFFFF" Duration="5.0" />
                    </Sequence>
                </OnUpdated>
            </Animations>
        </cc1:UpdatePanelAnimationExtender>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <asp:Label ID="Label1" runat="server"></asp:Label><br />
                <asp:Calendar ID="Calendar1" runat="server"
                   onselectionchanged="Calendar1_SelectionChanged"></asp:Calendar>
            </ContentTemplate>
        </asp:UpdatePanel>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<script runat="server">
    protected void Calendar1_SelectionChanged(object sender, EventArgs e)
    {
        Label1.Text = "The date selected is " +
           Calendar1.SelectedDate.ToLongDateString();
    }
</script>
```

With this bit of code, when you click a date within the Calendar control, the entire background of the UpdatePanel holding the calendar changes color from one to another for a five-second duration according as specified in the animation you built. The animations you define can get pretty complex, and building deluxe animations are beyond the scope of this chapter.

ValidatorCalloutExtender

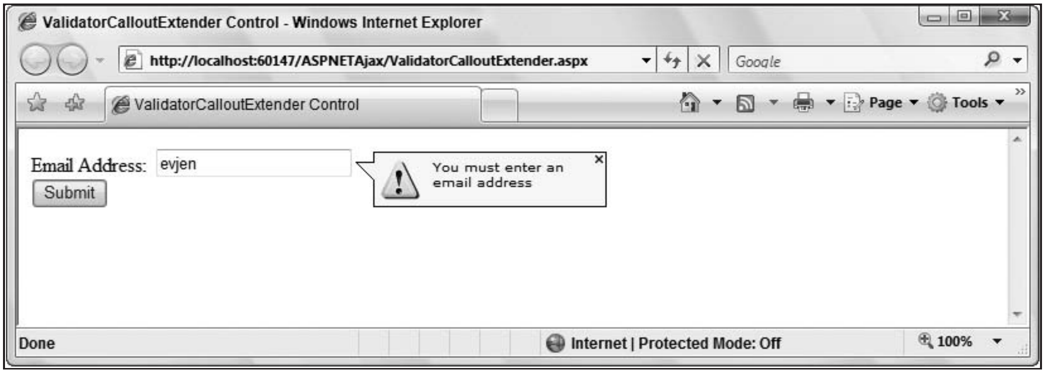
The last extender control covered is the ValidatorCalloutExtender control. This control allows you to add a more noticeable validation message to end users working with a form. You associate this control not with the control that is being validated, but instead with the validation control itself. An example of associating the ValidatorCalloutExtender control with a RegularExpressionValidator control is presented in Listing 20-34.

Listing 20-34: Creating validation callouts with the ValidationCalloutExtender

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>ValidatorCalloutExtender Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:ValidatorCalloutExtender ID="ValidatorCalloutExtender1" runat="server"
            TargetControlID="RegularExpressionValidator1">
        </cc1:ValidatorCalloutExtender>
        Email Address:&nbsp;
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <asp:RegularExpressionValidator ID="RegularExpressionValidator1"
         runat="server"
         ErrorMessage="You must enter an email address" Display="None"
         ControlToValidate="TextBox1"
         ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
        </asp:RegularExpressionValidator><br />
        <asp:Button ID="Button1" runat="server" Text="Submit" />
    </div>
    </form>
</body>
</html>
```

This page has a single text box for the form, a Submit button and a RegularExpressionValidator control. The RegularExpressionValidator control is built as you normally would, except you make use of the `Display` property and set it to `None`. You do not want the normal ASP.NET validation control to also display its message, as it will collide with the one displayed with the ValidatorCalloutExtender control. Although the `Display` property is set to `None`, you still use the `ErrorMessage` property to provide the error message. Running this page produces the results presented in Figure 20-41.



Email Address: evjen

You must enter an email address

ValidatorCalloutExtender Control - Windows Internet Explorer
http://localhost:60147/ASPNETAjax/ValidatorCalloutExtender.aspx
Google
ValidatorCalloutExtender Control
Page
Tools
Submit
Done
Internet | Protected Mode: Off
100%

Figure 20-41

ASP.NET AJAX Control Toolkit Server Controls

The next set of ASP.NET AJAX controls actually do not always extend other ASP.NET controls, but instead, are controls themselves. The following sections detail these controls.

Accordion Control

The `Accordion` control gives you the ability to provide a series of collapsible panes to the end user. This control is ideal when you have a lot of content to present in a limited space. The `Accordion` control is a template control and contains panes represented as AccordionPane controls. Listing 20-35 shows an Accordion control that contains two AccordionPane controls.

Listing 20-35: An Accordion control with two AccordionPane controls

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Accordion Control</title>
    <style type="text/css">
        .titlebar
        {
         background-color:Blue;
         color:White;
         font-size:large;
         font-family:Verdana;
         border:solid 3px Black;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:Accordion ID="Accordion1" runat="server" HeaderCssClass="titlebar"
         HeaderSelectedCssClass="titlebar">
            <Panes>
            <cc1:AccordionPane runat="server">
                <Header>
                    This is the first pane
                </Header>
                <Content>
            Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
            Donec accumsan lorem. Ut consectetuer tempus metus. Aenean tincidunt
            venenatis tellus. Suspendisse molestie cursus ipsum. Curabitur ut
            lectus. Nulla ac dolor nec elit convallis vulputate. Nullam pharetra
            pulvinar nunc. Duis orci. Phasellus a tortor at nunc mattis congue.
            Vestibulum porta tellus eu orci. Suspendisse quis massa. Maecenas
            varius, erat non ullamcorper nonummy, mauris erat eleifend odio, ut
            gravida nisl neque a ipsum. Vivamus facilisis. Cras viverra. Curabitur
            ut augue eget dolor semper posuere. Aenean at magna eu eros tempor
            pharetra. Aenean mauris.
                </Content>
            </cc1:AccordionPane>
```

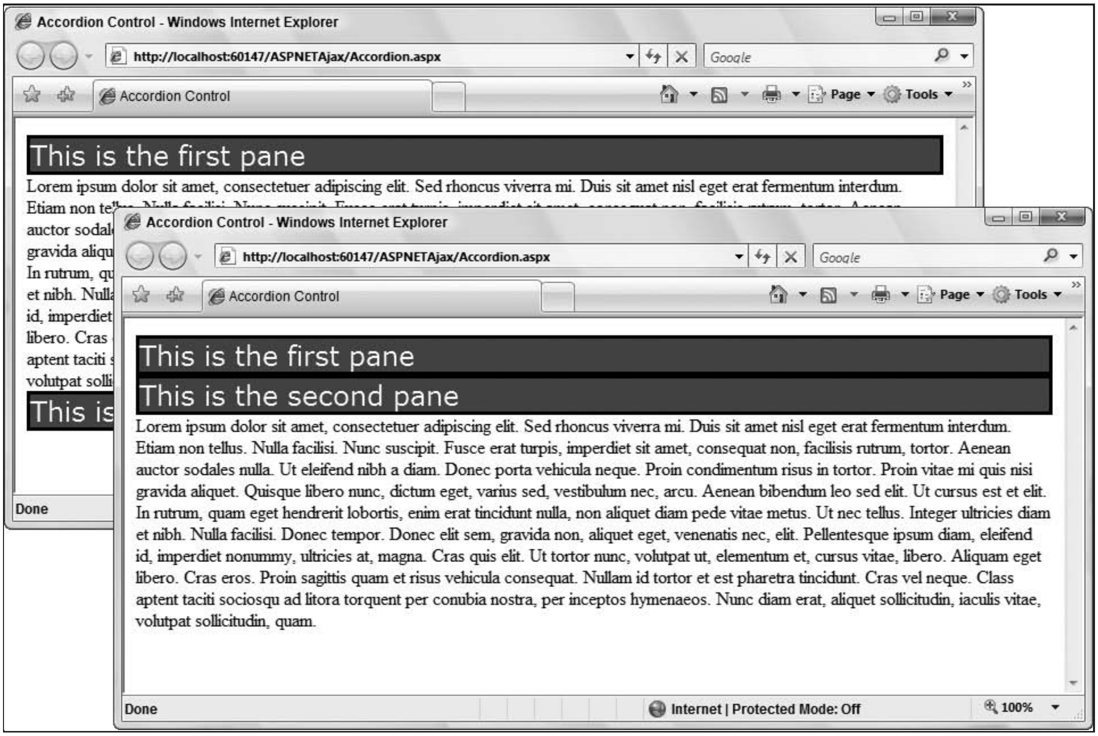
```
                <cc1:AccordionPane runat="server">
                    <Header>
                        This is the second pane
                    </Header>
                    <Content>
                Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
                Donec accumsan lorem. Ut consectetuer tempus metus. Aenean tincidunt
                venenatis tellus. Suspendisse molestie cursus ipsum. Curabitur ut
                lectus. Nulla ac dolor nec elit convallis vulputate. Nullam pharetra
                pulvinar nunc. Duis orci. Phasellus a tortor at nunc mattis congue.
                Vestibulum porta tellus eu orci. Suspendisse quis massa. Maecenas
                varius, erat non ullamcorper nonummy, mauris erat eleifend odio, ut
                gravida nisl neque a ipsum. Vivamus facilisis. Cras viverra. Curabitur
                ut augue eget dolor semper posuere. Aenean at magna eu eros tempor
                pharetra. Aenean mauris.
                    </Content>
                </cc1:AccordionPane>
                </Panes>
            </cc1:Accordion>
        </div>
        </form>
    </body>
    </html>
```

There is a single CSS class defined in the document and this class, `titlebar`, is used as the value of the `HeaderCssClass` and the `HeaderSelectedCssClass` properties. The Accordion control here contains two AccordionPane controls. The subelements of the AccordionPane control are the `<Header>` and the `<Content>` elements. The items placed in the `<Header>` section will be in the clickable pane title, while the items contained within the `<Content>` section will slide out and be presented when the associated header is selected.

Running this page produces the results illustrated here in Figure 20-42.

This figure shows a screenshot of each of the panes selected. Some of the more important properties are described in the following list:

- `AutoSize` — Defines how the control deals with its size expansion and shrinkage. The possible values include `None`, `Fill`, and `Limit`. The default is `None` and when used, items below the control may move to make room for the control expansion. A value of `Fill` works with the `Height` property and the control will fill to the required `Height`. This means that some of the panes may have to grow to accommodate the space while other panes might have to shrink and include a scrollbar to handle the limited space from the height restriction. A value of `Limit` also works with the `Height` property and will never grow larger than this value. It is possible that the pane might be smaller than the specified height.
- `TransitionDuration` — The number of milliseconds it takes to transition to another pane.
- `FramesPerSecond` — The number of frames per second to use to transition to another pane.
- `RequireOpenedPane` — Specifies that at least one pane is required to be open at all times. The default setting of this property is `True`. A value of `False` means that all panes can be collapsed.


Accordion Control - Windows Internet Explorer
http://localhost:60147/ASPNETAjax/Accordion.aspx
Google
Accordion Control
Page
Tools
Done
Internet | Protected Mode: Off
100%

This is the first pane

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed rhoncus viverra mi. Duis sit amet nisl eget erat fermentum interdum.

This is the first pane

This is the second pane

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed rhoncus viverra mi. Duis sit amet nisl eget erat fermentum interdum. Etiam non tellus. Nulla facilisi. Nunc suscipit. Fusce erat turpis, imperdiet sit amet, consequat non, facilisis rutrum, tortor. Aenean auctor sodales nulla. Ut eleifend nibh a diam. Donec porta vehicula neque. Proin condimentum risus in tortor. Proin vitae mi quis nisi gravida aliquet. Quisque libero nunc, dictum eget, varius sed, vestibulum nec, arcu. Aenean bibendum leo sed elit. Ut cursus est et elit. In rutrum, quam eget hendrerit lobortis, enim erat tincidunt nulla, non aliquet diam pede vitae metus. Ut nec tellus. Integer ultricies diam et nibh. Nulla facilisi. Donec tempor. Donec elit sem, gravida non, aliquet eget, venenatis nec, elit. Pellentesque ipsum diam, eleifend id, imperdiet nonummy, ultricies at, magna. Cras quis elit. Ut tortor nunc, volutpat ut, elementum et, cursus vitae, libero. Aliquam eget libero. Cras eros. Proin sagittis quam et risus vehicula consequat. Nullam id tortor et est pharetra tincidunt. Cras vel neque. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Nunc diam erat, aliquet sollicitudin, iaculis vitae, volutpat sollicitudin, quam.

Figure 20-42

Finally, the properties of `DataSource`, `DataSourceID`, and `DataMember` allow you to bind to this control from your code.

NoBot Control

The NoBot control works to determine how entities interact with your forms and to help you make sure that actual humans are working with your forms and some automated code isn't working through your application.

The NoBot control is illustrated in Listing 20-36.

Listing 20-36: Using the NoBot control to limit a login form

.ASPX

```
<%@ Page Language="VB" AutoEventWireup="true" CodeFile="NoBot.aspx.vb"
    Inherits="NoBot" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>NoBot Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:NoBot ID="NoBot1" runat="server" CutoffMaximumInstances="3"
            CutoffWindowSeconds="15" ResponseMinimumDelaySeconds="10"
            OnGenerateChallengeAndResponse="NoBot1_GenerateChallengeAndResponse" />
        <asp:Login ID="Login1" runat="server">
        </asp:Login>
        <asp:Label ID="Label1" runat="server"></asp:Label>
    </div>
    </form>
</body>
</html>
```

The NoBot control has three important properties to be aware of when controlling how your forms are submitted. These properties include the `CutoffMaximumInstances`, `CutoffWindowSeconds`, and the `ResponseMinimumDelaySeconds` properties.

The `CutoffMaximumInstances` is the number of times the end user is allowed to try to submit the form within the number of seconds specified by the `CutoffWindowSeconds` property. The `ResponseMinimumDelaySeconds` property defines the minimum number of seconds the end user has to submit the form. If you know the form you are working with will take some time, then setting this property to a value (even if it is `5` seconds) will help stop submissions that are not made by humans.

The `OnGenerateChallengeAndResponse` property allows you to define the server-side method that works with the challenge and allows you to provide a response based on the challenge. This property is used in Listing 20-36 and posts back to the user the status of the form submission.

The code-behind for this page is represented in Listing 20-37.

Listing 20-37: The code-behind page for the NoBot control's OnGenerateChallengeAndResponse

VB

```
Imports System
Imports AjaxControlToolkit

Public partial Class NoBot
 Inherits System.Web.UI.Page
    Protected Sub NoBot1_GenerateChallengeAndResponse(ByVal sender As Object, _
      ByVal void As AjaxControlToolkit.NoBotEventArgs) _
      Handles NoBot1.GenerateChallengeAndResponse
```

Continued

```
        Dim state As NoBotState
        NoBot1.IsValid(state)

        Label1.Text = state.ToString()
    End Sub
End Class
```

C#

```
using System;
using AjaxControlToolkit;

public partial class NoBot : System.Web.UI.Page
{
    protected void NoBot1_GenerateChallengeAndResponse(object sender,
        AjaxControlToolkit.NoBotEventArgs e)
    {
        NoBotState state;
        NoBot1.IsValid(out state);

        Label1.Text = state.ToString();
    }
}
```

Running this page and trying to submit the form before the ten-second minimum time results in an invalid submission. In addition, trying to submit the form more than three times within 15 seconds results in an invalid submission.

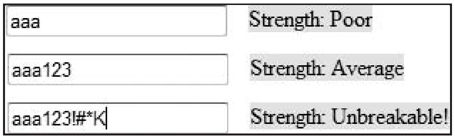
PasswordStrength Control

The PasswordStrength control allows you to check the contents of a password in a TextBox control and validate its strength. It will also then give a message to the end user about whether the strength is reasonable. A simple example of the PasswordStrength control is presented in Listing 20-38.

Listing 20-38: Using the PasswordStrength control with a TextBox control

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Password Strength Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:PasswordStrength ID="PasswordStrength1" runat="server"
         TargetControlID="TextBox1">
        </cc1:PasswordStrength>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </div>
    </form>
</body>
</html>
```

This simple page produces a single text box and when end users start typing in the text box, they will be notified on the strength of the submission as they type. This is illustrated in Figure 20-43.



aaa Strength: Poor

aaa123 Strength: Average

aaa123!#*K Strength: Unbreakable!

Figure 20-43

Some of the important properties to work with here include `MinimumLowerCaseCharacters`, `MinimumNumericCharacters`, `MinimumSymbolCharacters`, `MinimumUpperCaseCharacters`, and `PreferredPasswordLength`.

Rating Control

The Rating control gives your end users the ability to view and set ratings (such as star ratings). You have control over the number of ratings, the look of the filled ratings, the look of the empty ratings, and more. Listing 20-39 shows you a page that shows a five-star rating system that gives end users the ability to set the rating themselves.

Listing 20-39: A rating control that the end user can manipulate

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Rating Control</title>
```

Continued

```
        <style type="text/css">
            .ratingStar {
                font-size: 0pt;
                width: 13px;
                height: 12px;
                margin: 0px;
                padding: 0px;
                cursor: pointer;
                display: block;
                background-repeat: no-repeat;
            }

            .filledRatingStar {
                background-image: url(Images/FilledStar.png);
            }

            .emptyRatingStar {
                background-image: url(Images/EmptyStar.png);
            }

            .savedRatingStar {
                background-image: url(Images/SavedStar.png);
            }
        </style>
    </head>
    <body>
        <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <cc1:Rating ID="Rating1" runat="server" StarCssClass="ratingStar"
             WaitingStarCssClass="savedRatingStar"
             FilledStarCssClass="filledRatingStar" EmptyStarCssClass="emptyRatingStar">
            </cc1:Rating>
        </div>
        </form>
    </body>
    </html>
```

Here, the Rating control uses a number of CSS classes to define its look and feel in various states. In addition to the CSS class properties (`StarCssClass`, `WaitingStarCssClass`, `FilledStarCssClass`, and `EmptyCssClass`), you can also specify rating alignments, the number of rating items (the default is `5`), the width, the current rating, and more. The code presented in Listing 20-39 produces the results shown in Figure 20-44.

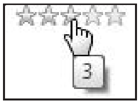


Figure 20-44

TabContainer Control

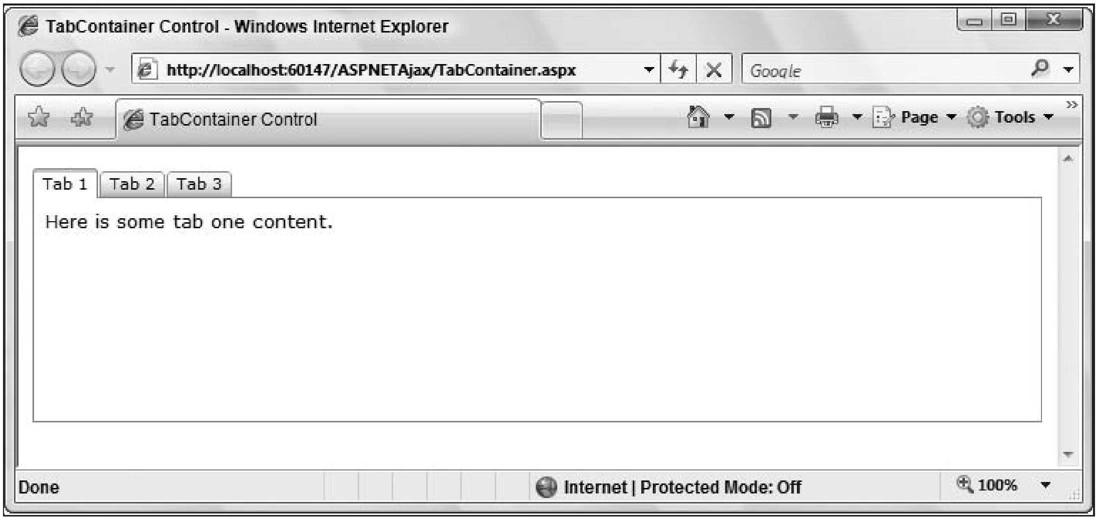
Tabs are another great way to control a page that has a lot of content to present. The `TabContainer` control can contain one or more TabPanel controls that provide you with a set of tabs that show content one tab at a time.

You are able to control the width and the height of the panels and to specify whether there are scrollbars as well. Each `TabPanel` control has `<HeaderTemplate>` and `<ContentTemplate>` subelement that you can define. Listing 20-40 shows an example of a TabContainer control with three TabPanel controls.

Listing 20-40: Showing three tabs in a TabContainer control

```
<%@ Page Language="C#" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
    TagPrefix="cc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>TabContainer Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <cc1:TabContainer ID="TabContainer1" runat="server" Height="300px">
            <cc1:TabPanel runat="server">
                <HeaderTemplate>Tab 1</HeaderTemplate>
                <ContentTemplate>Here is some tab one content.</ContentTemplate>
            </cc1:TabPanel>
            <cc1:TabPanel runat="server">
                <HeaderTemplate>Tab 2</HeaderTemplate>
                <ContentTemplate>Here is some tab two content.</ContentTemplate>
            </cc1:TabPanel>
            <cc1:TabPanel runat="server">
                <HeaderTemplate>Tab 3</HeaderTemplate>
                <ContentTemplate>Here is some tab three content.</ContentTemplate>
            </cc1:TabPanel>
        </cc1:TabContainer>
    </div>
    </form>
</body>
</html>
```

The result of this simple page is presented in Figure 20-45.


TabContainer Control - Windows Internet Explorer
http://localhost:60147/ASPNETAjax/TabContainer.aspx
Google
TabContainer Control
Page
Tools
Tab 1
Tab 2
Tab 3
Done
Internet | Protected Mode: Off
100%

Here is some tab one content.

Figure 20-45

Summary

As you can see, there are a ton of new controls at your disposal. The best thing about this is that this is a community effort along with Microsoft and the list of available ASP.NET AJAX controls is only going to grow over time.

This chapter looked at the lot of the new ASP.NET AJAX controls and how to use them in your ASP.NET applications. Remember to visit the CodePlex page for these controls often and take advantage of the newest offerings out there.

21

Security

Not every page that you build with ASP.NET is meant to be open and accessible to everyone on the Internet. Sometimes, you want to build pages or sections of an application that are accessible to only a select group of your choosing. For this reason, you need the security measures explained in this chapter. They can help protect the data behind your applications and the applications themselves from fraudulent use.

Security is a very wide-reaching term. During every step of the application-building process, you must, without a doubt, be aware of how mischievous end users might attempt to bypass your lockout measures. You must take steps to ensure that no one can take over the application or gain access to its resources. Whether it involves working with basic server controls or accessing databases, you should be thinking through the level of security you want to employ to protect yourself.

How security is applied to your applications is truly a measured process. For instance, a single ASP.NET page on the Internet, open to public access, has different security requirements than does an ASP.NET application that is available to only selected individuals because it deals with confidential information such as credit card numbers or medical information.

The first step is to apply the appropriate level of security for the task at hand. Because you can take so many different actions to protect your applications and the resources, you have to decide for yourself which of these measures to employ. This chapter looks at some of the possibilities for protecting your applications.

Notice that security is discussed throughout this book. In addition, a couple chapters focus on specific security frameworks provided by ASP.NET 3.5 that are not discussed in this chapter. Chapters 15 and 16 discuss ASP.NET's membership and role management frameworks, as well as the personalization features in this version. These topics are aspects of security that can make it even easier for you to build safe applications. Although these new security frameworks are provided with this latest release of ASP.NET, you can still build your own measures as you did in the previous versions of ASP.NET. This chapter discusses how to do so.

An important aspect of security is how you handle the authentication and authorization for accessing resources in your applications. Before you begin working through some of the authentication/authorization possibilities in ASP.NET, you should know exactly what we mean by those two terms.

Authentication and Authorization

As discussed in Chapter 16, *authentication* is the process that determines the identity of a user. After a user has been authenticated, a developer can determine if the identified user has authorization to proceed. It is impossible to give an entity authorization if no authentication process has been applied.

Authorization is the process of determining whether an authenticated user is permitted access to any part of an application, access to specific points of an application, or access only to specified datasets that the application provides. Authenticating and authorizing users and groups enable you to customize a site based on user types or preferences.

Applying Authentication Measures

ASP.NET provides many different types of authentication measures to use within your applications, including basic authentication, digest authentication, forms authentication, Passport, and Integrated Windows authentication. You also can develop your own authentication methods. You should never authorize access to resources you mean to be secure if you have not applied an authentication process to the requests for the resources.

The different authentication modes are established through settings that can be applied to the application's `web.config` file or in conjunction with the application server's Internet Information Services (IIS) instance.

ASP.NET is configured through a series of `.config` files on the application server. These are XML-based files that enable you to easily change how ASP.NET behaves. This is an ideal way to work with the configuration settings you require. ASP.NET configuration files are applied in a hierarchal manner. The .NET Framework provides a server-level configuration file called the `machine.config` file, which can be found at `C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG`. The folder contains the `machine.config` file. This file provides ASP.NET application settings at a server-level, meaning that the settings are applied to each and every ASP.NET application that resides on the particular server.

A `web.config` file is another XML-based configuration file that resides in the root of the Web application. The settings applied in the `web.config` file override the same settings applied in the higher-level `machine.config` file.

You can even nest the `web.config` files so that the main application `web.config` file is located in the root directory of your application, but additional `web.config` files reside in some of the application's subdirectories (see Figure 21-1). The `web.config` files contained in any of the subdirectories supersede the root directory's `web.config` file. Therefore, any settings applied through a subdirectory's `web.config` file change whatever was set in the application's main `web.config` file.

In many of the examples in this chapter, you use the `web.config` file to apply the authentication and authorization mechanics you want in your applications. You also can work with IIS to apply settings directly to your applications.

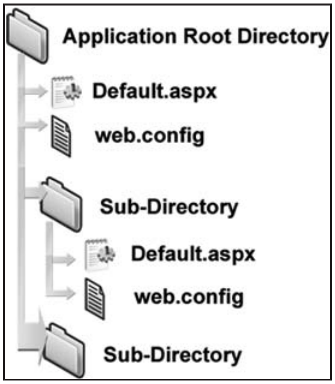


Figure 21-1

IIS is the Web server that handles all the incoming HTTP requests that come into the server. You must modify IIS to perform as you want. IIS hands a request to the ASP.NET engine only if the page has a specific file extension (for example, `.aspx`). In this chapter, you will work with IIS 7.0, as well.

The <authentication> Node

You use the `<authentication>` node in the application's `web.config` file to set the type of authentication your ASP.NET application requires:

```
<system.web>
   <authentication mode="Windows|Forms|Passport|None">

   </authentication>
</system.web>
```

The `<authentication>` node uses the `mode` attribute to set the form of authentication that is to be used. Options include `Windows`, `Forms`, `Passport`, and `None`. Each option is explained in the following table.

Provider	Description
`Windows`	Windows authentication is used together with IIS authentication. Authentication is performed by IIS in the following ways: basic, digest, or Integrated Windows Authentication. When IIS authentication is complete, ASP.NET uses the authenticated identity to authorize access. This is the default setting.
`Forms`	Requests that are not authenticated are redirected to an HTML form using HTTP client-side redirection. The user provides his login information and submits the form. If the application authenticates the request, the system issues a form that contains the credentials or a key for reacquiring the identity.
`Passport`	A centralized authentication service provided by Microsoft that offers single login and core profile services for member sites. This mode of authentication was de-emphasized by Microsoft at the end of 2004.
`None`	No authentication mode is in place with this setting.

As you can see, a couple of methods are at your disposal for building an authentication/authorization model for your ASP.NET applications. We start by examining the Windows mode of authentication.

Windows-Based Authentication

Windows-based authentication is handled between the Windows server where the ASP.NET application resides and the client machine. In a Windows-based authentication model, the requests go directly to IIS to provide the authentication process. This type of authentication is quite useful in an intranet environment where you can let the server deal completely with the authentication process — especially in environments where users are already logged onto a network. In this scenario, you simply grab and utilize the credentials that are already in place for the authorization process.

IIS first takes the user's credentials from the domain login. If this process fails, IIS displays a pop-up dialog box so the user can enter or re-enter his login information. To set up your ASP.NET application to work with Windows-based authentication, begin by creating some users and groups.

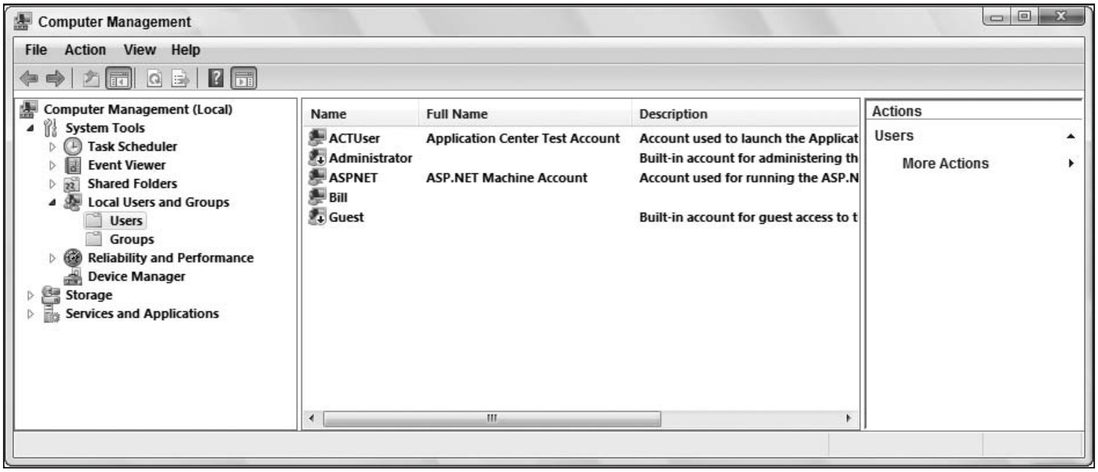
Creating Users

You use aspects of Windows-based authentication to allow specific users who have provided a domain login to access your application or parts of your application. Because it can use this type of authentication, ASP.NET makes it quite easy to work with applications that are deployed in an intranet environment. If a user has logged onto a local computer as a domain user, he will not need to be authenticated again when accessing a network computer in that domain.

The following steps show you how to create a user. It is important to note that you must have sufficient rights to be authorized to create users on a server. If you are authorized, the steps to create users are as follows:

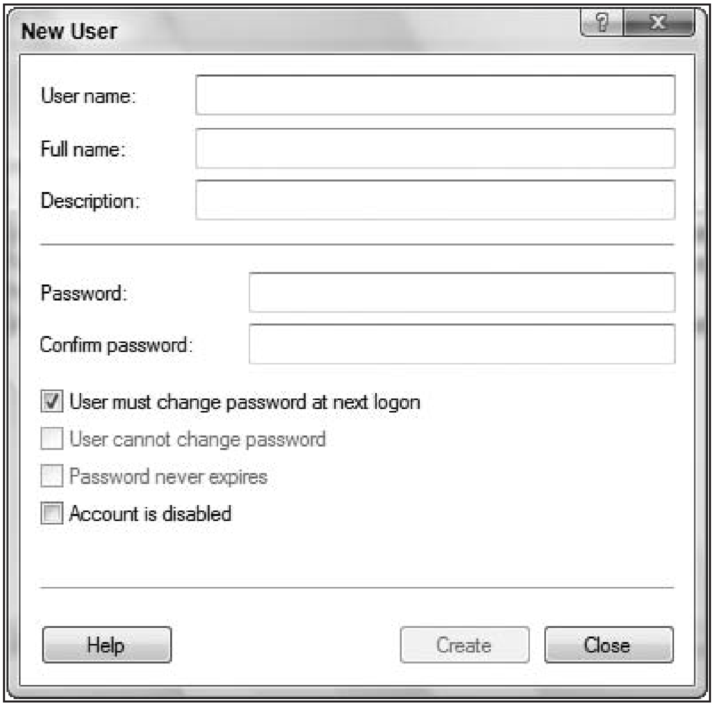
1. Within your Windows XP or Windows Server 2003 server, choose Start ➪ Control Panel ➪ Administrative Tools ➪ Computer Management. If you are using Windows Vista, choose Start ➪ Control Panel ➪ System and Maintenance ➪ Administrative Tools ➪ Computer Management. Either one opens the Computer Management utility. It manages and controls resources on the local Web server. You can accomplish many things using this utility, but the focus here is on the creation of users.
2. Expand the System Tools node.
3. Expand the Local Users and Groups node.
4. Select the Users folder. You see something similar to the results shown in Figure 21-2.
5. Right-click the Users folder and select New User. The New User dialog appears, as shown in Figure 21-3.
6. Give the user a name, password, and description stating that this is a test user. In this example, the user is called **Bubbles**.
7. Clear the check box that requires the user to change his password at the next login.
8. Click the Create button. Your test user is created and presented in the Users folder of the Computer Management utility, as shown in Figure 21-4.

Now create a page to work with this user.


Computer Management
File Action View Help
Computer Management (Local)
System Tools
Task Scheduler
Event Viewer
Shared Folders
Local Users and Groups
Users
Groups
Reliability and Performance
Device Manager
Storage
Services and Applications
Actions
Users
More Actions

Name	Full Name	Description
ACTUser	Application Center Test Account	Account used to launch the Applicat
Administrator		Built-in account for administering th
ASPNET	ASP.NET Machine Account	Account used for running the ASP.N
Bill		
Guest		Built-in account for guest access to t

Figure 21-2


New User
Help
Create
Close

User name:

Full name:

Description:

Password:

Confirm password:

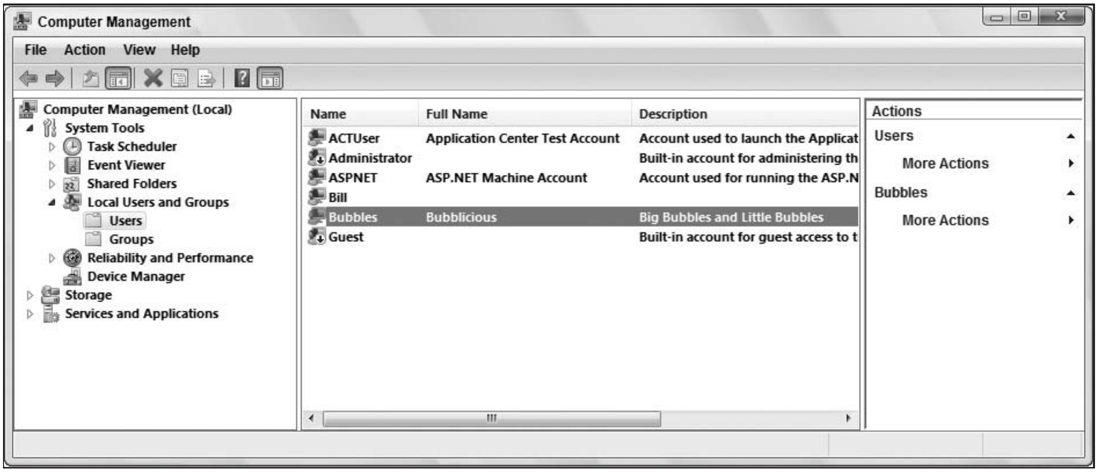
☑ User must change password at next logon
☐ User cannot change password
☐ Password never expires
☐ Account is disabled

Figure 21-3

Authenticating and Authorizing a User

Now create an application that enables the user to enter it. You work with the application's `web.config` file to control which users are allowed to access the site and which users are not allowed.

Add the section presented in Listing 21-1 to your `web.config` file.


Computer Management
File
Action
View
Help
Computer Management (Local)
System Tools
Task Scheduler
Event Viewer
Shared Folders
Local Users and Groups
Users
Groups
Reliability and Performance
Device Manager
Storage
Services and Applications
Actions
Users
More Actions
Bubbles
More Actions

Name	Full Name	Description
ACTUser	Application Center Test Account	Account used to launch the Applicat
Administrator		Built-in account for administering th
ASPNET	ASP.NET Machine Account	Account used for running the ASP.N
Bill		
Bubbles	Bubblicious	Big Bubbles and Little Bubbles
Guest		Built-in account for guest access to t

Figure 21-4

Listing 21-1: Denying all users through the web.config file

```
<system.web>
   <authentication mode="Windows" />
   <authorization>
      <deny users="*" />
   </authorization>
</system.web>
```

In this example, the `web.config` file is configuring the application to employ Windows-based authentication using the `<authentication>` element's mode attribute. In addition, the `<authorization>` element is used to define specifics about the users or groups who are permitted access to the application. In this case, the `<deny>` element specifies that all users (even if they are authenticated) are denied access to the application. Not permitting specific users with the `<allow>` element does not make much sense, but for this example, leave it as it is. The results are illustrated in Figure 21-5.

Any end user — authenticated or not — who tries to access the site sees a large ''Access is denied'' statement in his browser window, which is just what you want for those not allowed to access your application!

In most instances, however, you want to allow at least some users to access your application. Use the `<allow>` element in the `web.config` file to allow a specific user. Here is the syntax:

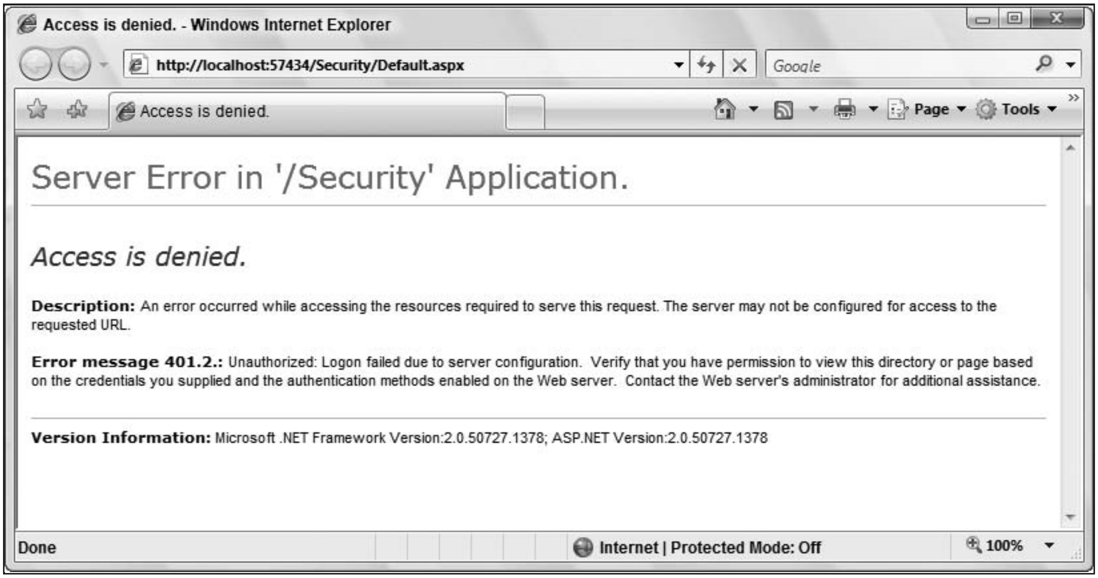
```
<allow users="Domain\Username" />
```

Listing 21-2 shows how the user is permitted access.

Listing 21-2: Allowing a single user through the web.config file

```
<system.web>
   <authentication mode="Windows" />
   <authorization>
```

```
      <allow users="REUTERS-EVJEN\Bubbles"/>
      <deny users="*"/>
   </authorization>
</system.web>
```


Access is denied. - Windows Internet Explorer
http://localhost:57434/Security/Default.aspx
Google
Access is denied.
Page
Tools
Done
Internet | Protected Mode: Off
100%

Server Error in '/Security' Application.

Access is denied.

Description: An error occurred while accessing the resources required to serve this request. The server may not be configured for access to the requested URL.

Error message 401.2.: Unauthorized: Logon failed due to server configuration. Verify that you have permission to view this directory or page based on the credentials you supplied and the authentication methods enabled on the Web server. Contact the Web server's administrator for additional assistance.

Version Information: Microsoft .NET Framework Version:2.0.50727.1378; ASP.NET Version:2.0.50727.1378

Figure 21-5

Even though all users (even authenticated ones) are denied access through the use of the `<deny>` element, the definitions defined in the `<allow>` element take precedence. In this example, a single user — `Bubbles` — is allowed.

Now, if you are logged on to the client machine as the user Bubbles and run the page in the browser, you get access to the application.

Looking Closely at the <allow> and <deny> Nodes

The `<allow>` and `<deny>` nodes enable you to work not only with specific users, but also with groups. The elements support the attributes defined in the following table.

Attribute	Description
`Users`	Enables you to specify users by their domain and/or name.
`Roles`	Enables you to specify access groups that are allowed or denied access.
`Verbs`	Enables you to specify the HTTP transmission method that is allowed or denied access.

When using any of these attributes, you can specify all users with the use of the asterisk (*):

```
<allow roles="*" />
```

In this example, all roles are allowed access to the application. Another symbol you can use with these attributes is the question mark (`?`), which represents all anonymous users. For example, if you want to block all anonymous users from your application, use the following construction:

```
<deny users="?" />
```

When using `users`, `roles`, or `verbs` attributes with the `<allow>` or `<deny>` elements, you can specify multiple entries by separating the values with a comma. If you are going to allow more than one user, you can either separate these users into different elements, as shown here:

```
<allow users="MyDomain\User1" />
<allow users="MyDomain\User2" />
```

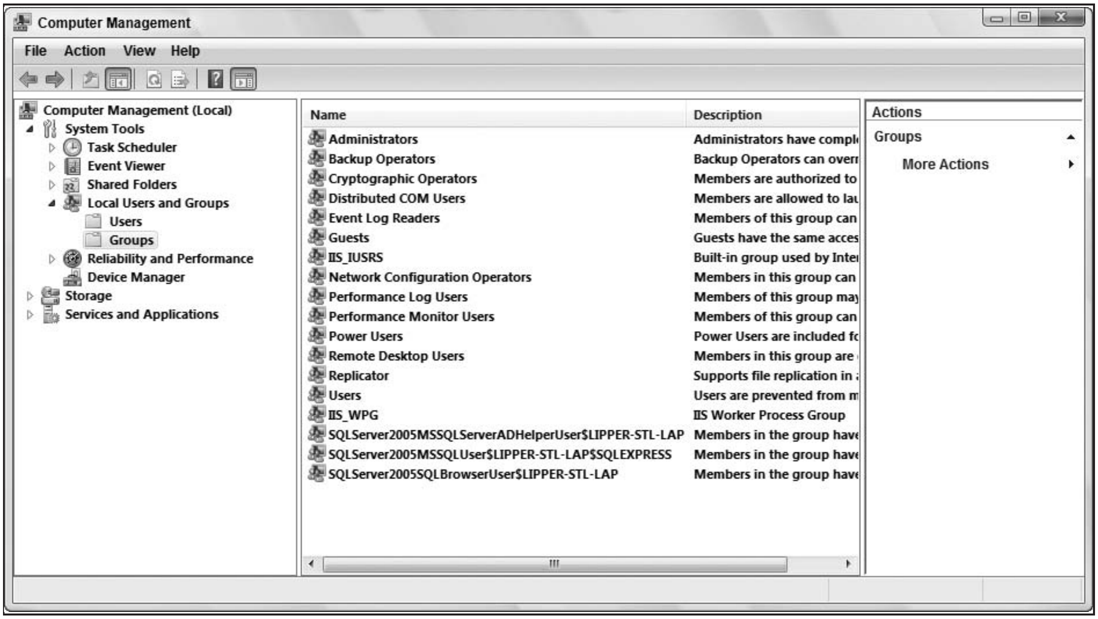
or you can use the following:

```
<allow users="MyDomain\User1, MyDomain\User2" />
```

Use the same construction when defining multiple roles and verbs.

Authenticating and Authorizing a Group

You can define groups of individuals allowed or denied access to your application or the application's resources. Your server can contain a number of different groups, each of which can have any number of users belonging to it. It is also possible for a single user to belong to multiple groups. Pull up the Computer Management utility to access the list of the groups defined on the server you are working with. Simply click the Groups folder in the Computer Management utility, and the list of groups is displayed, as illustrated in Figure 21-6.


Computer Management
File
Action
View
Help
Computer Management (Local)
System Tools
Task Scheduler
Event Viewer
Shared Folders
Local Users and Groups
Users
Groups
Reliability and Performance
Device Manager
Storage
Services and Applications
Actions
Groups
More Actions

Name	Description
Administrators	Administrators have compl
Backup Operators	Backup Operators can overr
Cryptographic Operators	Members are authorized to
Distributed COM Users	Members are allowed to lau
Event Log Readers	Members of this group can
Guests	Guests have the same acces
IIS_IUSRS	Built-in group used by Inte
Network Configuration Operators	Members in this group can
Performance Log Users	Members of this group may
Performance Monitor Users	Members of this group can
Power Users	Power Users are included fo
Remote Desktop Users	Members in this group are
Replicator	Supports file replication in
Users	Users are prevented from m
IIS_WPG	IIS Worker Process Group
SQLServer2005MSSQLServerADHelperUser$LIPPER-STL-LAP	Members in the group have
SQLServer2005MSSQLUser$LIPPER-STL-LAP$SQLEXPRESS	Members in the group have
SQLServer2005SQLBrowserUser$LIPPER-STL-LAP	Members in the group have

Figure 21-6

Right-click in the Groups folder to select New Group. The New Group dialog displays (see Figure 21-7).

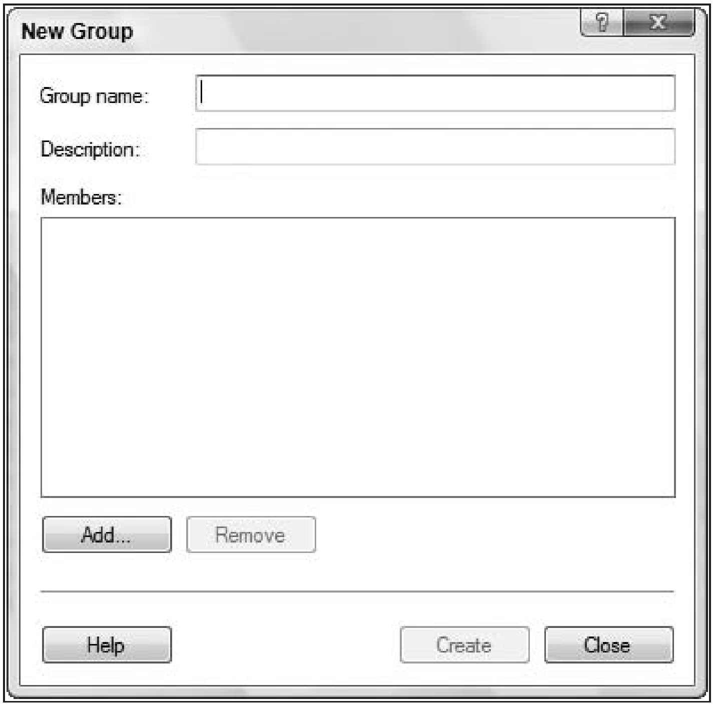


Figure 21-7

To create a group, give it a name and description; then click the Add button and select the users whom you want to be a part of the group. After a group is created, you can allow it access to your application like this:

```
<allow roles="MyGroup" />
```

You can use the `roles` attribute in either the `<allow>` or `<deny>` element to work with a group that you have created or with a specific group that already exists.

Authenticating and Authorizing an HTTP Transmission Method

In addition to authenticating and authorizing specific users or groups of users, you can also authorize or deny requests that come via a specific HTTP transmission protocol. This is done using the `verb` attribute in the `<allow>` and `<deny>` elements.

```
<deny verbs="GET, DEBUG" />
```

In this example, requests that come in using the HTTP GET or HTTP DEBUG protocols are denied access to the site. Possible values for the `verbs` attribute include `POST`, `GET`, `HEAD`, and `DEBUG`.

Integrated Windows Authentication

So far, you have been using the default Integrated Windows authentication mode for the authentication/authorization process. This is fine if you are working with an intranet application and each of the clients is using Windows, the only system that the authentication method supports. This system of authentication also requires the client to be using Microsoft's Internet Explorer, which might not always be possible.

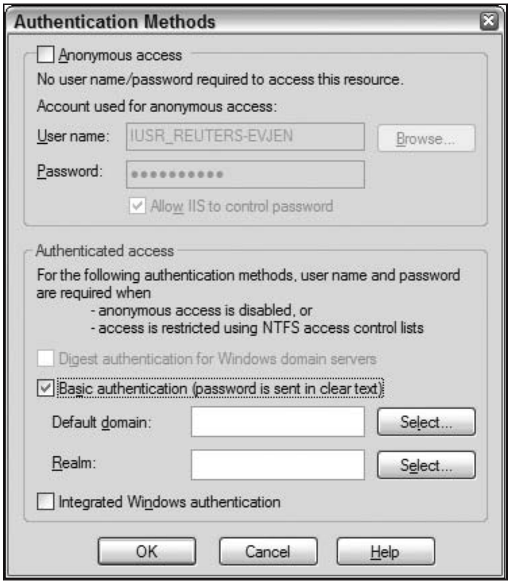
Integrated Windows authentication was previously known as NTLM or Windows NT Challenge/Response authentication. This authentication model has the client prove its identity by sending a hash of its credentials to the server that is hosting the ASP.NET application. Along with Microsoft's Active Directory, a client can also use Kerberos if it is using Microsoft's Internet Explorer 5 or higher.

Basic Authentication

Another option is to use Basic authentication, which also requires a username and password from the client for authentication. The big plus about Basic authentication is that it is part of the HTTP specification and therefore is supported by most browsers. The negative aspect of Basic authentication is that it passes the username and password to the server as clear text, meaning that the username and password are quite visible to prying eyes. For this reason, it is important to use Basic authentication along with SSL (*Secure Sockets Layer*).

If you are using IIS 5 or 6, to implement Basic authentication for your application, you must pull up IIS and open the Properties dialog for the Web site you are working with. Select the Directory Security tab and click the Edit button in the Anonymous Access and Authentication Control box. The Authentication Methods dialog box opens.

Uncheck the Integrated Windows Authentication check box at the bottom and check the Basic Authentication check box above it (see Figure 21-8). When you do, you are warned that this method transmits usernames and passwords as clear text.



Authentication Methods

☐ Anonymous access

No user name/password required to access this resource.

Account used for anonymous access:

User name: IUSR_REUTERS-EVJEN

Password: ••••••••••

☑ Allow IIS to control password

Authenticated access

For the following authentication methods, user name and password are required when
- anonymous access is disabled, or
- access is restricted using NTFS access control lists

☐ Digest authentication for Windows domain servers

☑ Basic authentication (password is sent in clear text)

Default domain:

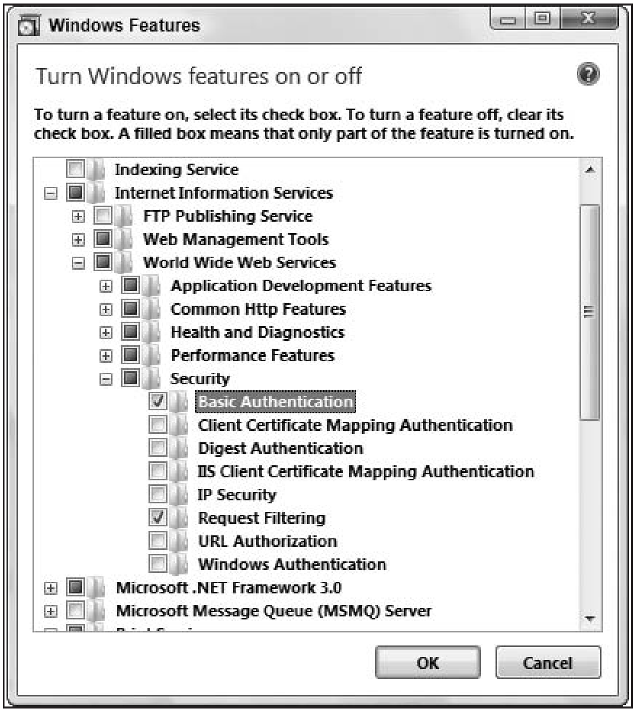
Realm:

☐ Integrated Windows authentication

Figure 21-8

End by clicking OK in the dialog. Now your application uses Basic authentication instead of Integrated Windows authentication.

If you are using Windows Vista, it is not easy to find the option to enable Basic authentication. Instead, you first have to enable IIS 7 to use Basic authentication by selecting Start ➪ Control Panel ➪ Programs ➪ Programs and Features ➪ Turn Windows features on or off. From the provided dialog box, navigate to the Internet Information Services section and expand until you arrive at World Wide Web Services ➪ Security. From here, check the Basic Authentication option and press OK to install. This option is presented in Figure 21-9.



To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- ☐ Indexing Service
- ■ Internet Information Services
  - ☐ FTP Publishing Service
  - ■ Web Management Tools
  - ■ World Wide Web Services
    - ■ Application Development Features
    - ■ Common Http Features
    - ■ Health and Diagnostics
    - ■ Performance Features
    - ■ Security
      - ☑ Basic Authentication
      - ☐ Client Certificate Mapping Authentication
      - ☐ Digest Authentication
      - ☐ IIS Client Certificate Mapping Authentication
      - ☐ IP Security
      - ☑ Request Filtering
      - ☐ URL Authorization
      - ☐ Windows Authentication
- ■ Microsoft .NET Framework 3.0
- ☐ Microsoft Message Queue (MSMQ) Server

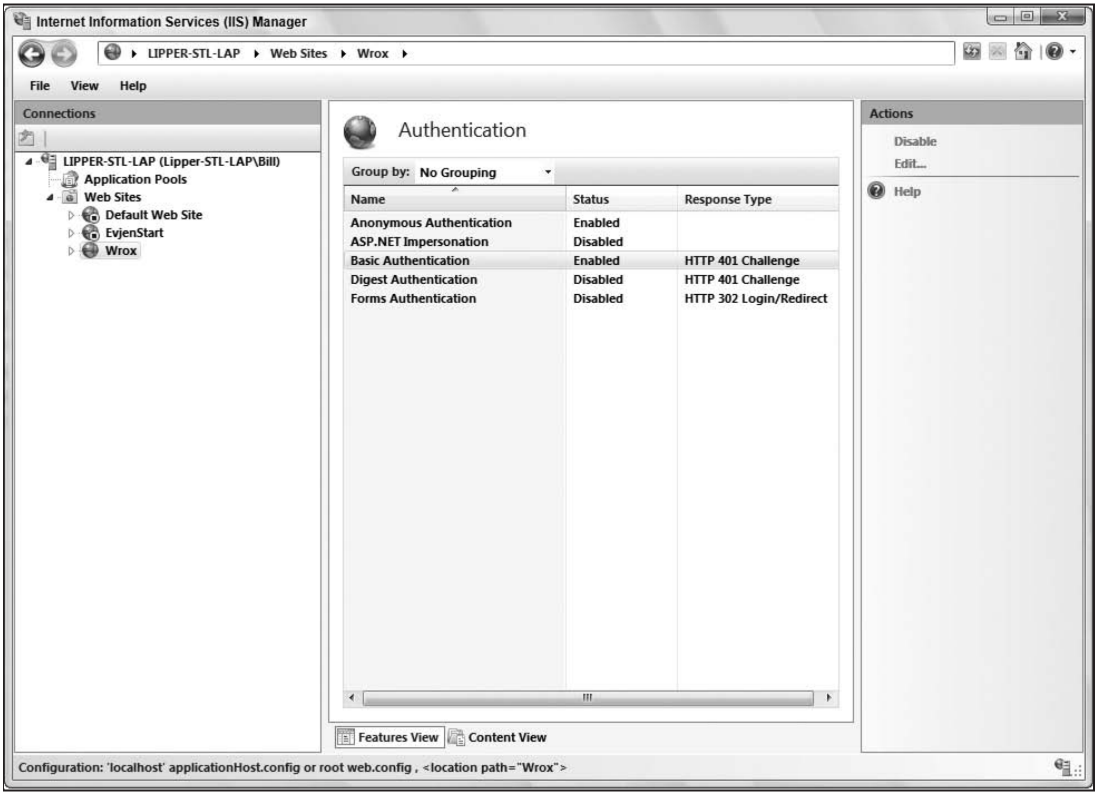
Figure 21-9

Once this option is installed, you can then return to the Internet Information Services (IIS) Manager and select the Authentication option in the IIS section for the virtual directory you are focusing on. From there, highlight the Basic Authentication option and select Enable from the Actions pane. This is illustrated in Figure 21-10.

Digest Authentication

Digest authentication is the final mode you explore in this chapter. The model alleviates the Basic authentication problem of passing the client's credentials as clear text. Instead, Digest authentication uses an algorithm to encrypt the client's credentials before they are sent to the application server.

To use Digest authentication, you are required to have a Windows domain controller. One of the main issues that arises with Digest authentication is that it is not supported on all platforms and requires


Internet Information Services (IIS) Manager
LIPPER-STL-LAP ▸ Web Sites ▸ Wrox ▸
File
View
Help
Connections
LIPPER-STL-LAP (Lipper-STL-LAP\Bill)
Application Pools
Web Sites
Default Web Site
EvjenStart
Wrox
Authentication
Actions
Disable
Edit...
Help
Features View
Content View

Group by: No Grouping

Name	Status	Response Type
Anonymous Authentication	Enabled	
ASP.NET Impersonation	Disabled	
Basic Authentication	Enabled	HTTP 401 Challenge
Digest Authentication	Disabled	HTTP 401 Challenge
Forms Authentication	Disabled	HTTP 302 Login/Redirect

Configuration: 'localhost' applicationHost.config or root web.config , <location path="Wrox">

Figure 21-10

browsers that conform to the HTTP 1.1 specification. Digest authentication, however, not only works well with firewalls, but it is also compatible with proxy servers.

You can select Digest authentication as the choice for your application in the same Authentication Methods dialog — simply select the Digest Authentication check box from the properties dialog if you are using IIS 5 or 6. If you are using IIS 7, you need to install Digest Authentication just as you installed Basic Authentication. Once installed, you will find this option and will be able to enable it from the Authentication section within the IIS Manager.

Forms-Based Authentication

Forms-based authentication is a popular mode of authenticating users to access an entire application or specific resources within an application. Using it enables you to put the login form directly in the application so that the end user simply enters his username and password into an HTML form contained within the browser itself. One negative aspect of forms-based authentication is that the usernames and passwords are sent as clear text unless you are using SSL.

It is easy and relatively straightforward to implement forms-based authentication in your Web application. To begin with, you make some modifications to your application's `web.config` file, as illustrated in Listing 21-3.

Listing 21-3: Modifying the web.config file for forms-based authentication

```
<system.web>
   <authentication mode="Forms">
      <forms name="Wrox" loginUrl="Login.aspx" path="/" />
   </authentication>

   <authorization>
      <deny users="?" />
   </authorization>
</system.web>
```

You must apply this structure to the `web.config` file. First, using the `<authorization>` element described earlier, you are denying access to the application to all anonymous users. Only authenticated users are allowed to access any page contained within the application.

If the requestor is not authenticated, what is defined in the `<authentication>` element is put into action. The value of the `mode` attribute is set to `Forms` to employ forms-based authentication for your Web application. The next attribute specified is `loginUrl`, which points to the page that contains the application's login form. In this example, `Login.aspx` is specified as a value. If the end user trying to access the application is not authenticated, his request is redirected to `Login.aspx` so that the user can be authenticated and authorized to proceed. After valid credentials have been provided, the user is returned to the location in the application where he originally made the request. The final attribute used here is `path`. It simply specifies the location in which to save the cookie used to persist the authorized user's access token. In most cases, you want to leave the value as `/`. The following table describes each of the possible attributes for the `<forms>` element.

Attribute	Description
`name`	This name is assigned to the cookie saved in order to remember the user from request to request. The default value is `.ASPXAUTH`.
`loginUrl`	Specifies the URL to which the request is redirected for login if no valid authentication cookie is found. The default value is `Login.aspx`.
`protection`	Specifies the amount of protection you want to apply to the authentication cookie. The four available settings are: ❑ `All`: The application uses both data validation and encryption to protect the cookie. This is the default setting. ❑ `None`: Applies no encryption to the cookie. ❑ `Encryption`: The cookie is encrypted but data validation is not performed on it. Cookies used in this manner might be subject to plain text attacks. ❑ `Validation`: The opposite of the `Encryption` setting. Data validation is performed, but the cookie is not encrypted.
`path`	Specifies the path for cookies issued by the application. In most cases you want to use `/`, which is the default setting.

Attribute	Description
`timeout`	Specifies the amount of time, in minutes, after which the cookie expires. The default value is `30`.
`cookieless`	Specifies whether the forms-based authentication process should use cookies when working with the authentication/authorization process.
`defaultUrl`	Specifies the default URL.
`domain`	Specifies the domain name to be sent with forms authentication cookies.
`slidingExpiration`	Specifies whether to apply a sliding expiration to the cookie. If set to `True`, the expiration of the cookie is reset with each request made to the server. The default value is `False`.
`enableCrossAppsRedirect`	Specifies whether to allow for cross-application redirection.
`requireSSL`	Specifies whether a Secure Sockets Layer (SSL) connection is required when transmitting authentication information.

After the `web.config` file is in place, the next step is to create a typical page for your application that people can access. Listing 21-4 presents a simple page.

Listing 21-4: A simple page — Default.aspx

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>The Application</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
       Hello World
    </div>
    </form>
</body>
</html>
```

As you can see, this page simply writes `Hello World` to the browser. The real power of forms authentication is shown in the `Login.aspx` page presented in Listing 21-5.

Listing 21-5: The Login.aspx page

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)
```

```
            If (TextBox1.Text = "BillEvjen" And TextBox2.Text = "Bubbles") Then
                FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, True)
            Else
                Response.Write("Invalid credentials")
            End If
        End Sub
    </script>

    <html xmlns="http://www.w3.org/1999/xhtml">
    <head runat="server">
        <title>Login Page</title>
    </head>
    <body>
        <form id="form1" runat="server">
        <div>
            Username<br />
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
            <br />
            Password<br />
            <asp:TextBox ID="TextBox2" runat="server"
             TextMode="Password"></asp:TextBox><br />
            <br />
            <asp:Button ID="Button1" OnClick="Button1_Click" runat="server"
             Text="Submit" />
        </div>
        </form>
    </body>
    </html>
```

C#

```
<%@ Page Language="C#"%>

<script runat="server">
   protected void Button1_Click(object sender, EventArgs e)
    {
        if (TextBox1.Text == "BillEvjen" && TextBox2.Text == "Bubbles") {
            FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, true);
        }
        else {
            Response.Write("Invalid credentials");
        }
    }
</script>
```

`Login.aspx` has two simple TextBox controls and a Button control that asks the user to submit his username and password. The `Button1_Click` event uses the `RedirectFromLoginPage` method of the `FormsAuthentication` class. This method does exactly what its name implies — it redirects the request from `Login.aspx` to the original requested resource.

`RedirectFromLoginPage` takes two arguments. The first is the name of the user, used for cookie authentication purposes. This argument does not actually map to an account name and is used by ASP.NET's URL authorization capabilities. The second argument specifies whether a durable cookie should be issued. If set to `True`, the end user does not need to log in again to the application from one browser session to the next.

Using the three pages you have constructed, each request for the `Default.aspx` page from Listing 21-4 causes ASP.NET to check that the proper authentication token is in place. If the proper token is not found, the request is directed to the specified login page (in this example, `Login.aspx`). Looking at the URL in the browser, you can see that ASP.NET is using a querystring value to remember where to return the user after he has been authorized to proceed:

```
http://localhost:35089/Security/Login.aspx?ReturnUrl=%2fSecurity%2fDefault.aspx
```

Here, the querystring `ReturnUrl` is used with a value of the folder and page that was the initial request.

Look more closely at the `Login.aspx` page from Listing 21-5, and note that the values placed in the two text boxes are checked to make sure they abide by a specific username and password. If they do, the `RedirectFromLoginPage` method is invoked; otherwise, the `Response.Write()` statement is used. In most cases, you do not want to hardcode a username and password in your code. Many other options exist for checking whether usernames and passwords come from authorized users. Some of the other options follow.

Authenticating Against Values Contained in the web.config File

The previous example is not the best approach for dealing with usernames and passwords offered for authentication. It is never a good idea to hardcode these things directly into your applications. Take a quick look at storing these values in the `web.config` file itself.

The `<forms>` element in `web.config` that you worked with in Listing 21-3 can also take a sub-element. The sub-element, `<credentials>`, allows you to specify username and password combinations directly in the `web.config` file. You can choose from a couple of ways to add these values. The simplest method is shown in Listing 21-6.

Listing 21-6: Modifying the web.config file to add username/password values

```
<system.web>
   <authentication mode="Forms">
      <forms name="Wrox" loginUrl="Login.aspx" path="/">
         <credentials passwordFormat="Clear">
            <user name="BillEvjen" password="Bubbles" />
         </credentials>
      </forms>
   </authentication>

   <authorization>
      <deny users="?" />
   </authorization>
</system.web>
```

The `<credentials>` element has been included to add users and their passwords to the configuration file. `<credentials>` takes a single attribute — `passwordFormat`. The possible values of `passwordFormat` are `Clear`, `MD5`, and `SHA1`. The following list describes each of these options:

- `Clear`: Passwords are stored in clear text. The user password is compared directly to this value without further transformation.
- `MD5`: Passwords are stored using a Message Digest 5 (MD5) hash digest. When credentials are validated, the user password is hashed using the MD5 algorithm and compared for equality with

this value. The clear-text password is never stored or compared. This algorithm produces better performance than SHA1.

- ❑ `SHA1`: Passwords are stored using the SHA1 hash digest. When credentials are validated, the user password is hashed using the SHA1 algorithm and compared for equality with this value. The clear-text password is never stored or compared. Use this algorithm for best security.

In the example from Listing 21-6, you use a setting of `Clear`. This is not the most secure method, but it is used for demonstration purposes. A sub-element of `<credentials>` is `<user>`; that is where you define the username and password for the authorized user with the attributes `name` and `password`.

The next step is to change the `Button1_Click` event on the `Login.aspx` page shown earlier. This is illustrated in Listing 21-7.

Listing 21-7: Changing the Login.aspx page to work with the web.config file

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)

        If FormsAuthentication.Authenticate(TextBox1.Text, TextBox2.Text) Then
            FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, True)
        Else
            Response.Write("Invalid credentials")
        End If
    End Sub
</script>
```

C#

```
<%@ Page Language="C#"%>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        if (FormsAuthentication.Authenticate(TextBox1.Text, TextBox2.Text)) {
            FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, true);
        }
        else {
            Response.Write("Invalid credentials");
        }
    }
</script>
```

In this example, you simply use the `Authenticate()` method to get your ASP.NET page to look at the credentials stored in the `web.config` file for verification. The `Authenticate()` method takes two parameters — the username and the password that you are passing in to be checked. If the credential lookup is successful, the `RedirectFromLoginPage` method is invoked.

It is best not to store your users' passwords in the `web.config` file as clear text as the preceding example did. Instead, use one of the available hashing capabilities so you can keep the end user's password out of sight of prying eyes. To do this, simply store the hashed password in the configuration file as shown in Listing 21-8.

Listing 21-8: Using encrypted passwords

```
<forms name="Wrox" loginUrl="Login.aspx" path="/">
   <credentials passwordFormat="SHA1">
      <user name="BillEvjen" password="58356FB4CAC0B801F011B397F9DFF45ADB863892" />
   </credentials>
</forms>
```

Using this kind of construct makes it impossible for even the developer to discover a password because the clear text password is never used. The `Authenticate()` method in the `Login.aspx` page hashes the password using SHA1 (because it is the method specified in the `web.config`'s `<credentials>` node) and compares the two hashes for a match. If a match is found, the user is authorized to proceed.

When using SHA1 or MD5, the only changes you make are in the `web.config` file and nowhere else. You do not have to make any changes to the login page or to any other page in the application. To store hashed passwords, however, you use the `FormsAuthenticationHashPasswordForStoringInConfigFile` method (probably the longest method name in the .NET Framework). You accomplish this in the following manner:

```
FormsAuthentication.HashPasswordForStoringInConfigFile(TextBox2.Text, "SHA1")
```

Authenticating Against Values in a Database

Another common way to retrieve username/password combinations is by getting them directly from a datastore of some kind. This enables you, for example, to check the credentials input by a user against values stored in Microsoft's SQL Server. The code for this is presented in Listing 21-9.

Listing 21-9: Checking credentials in SQL Server (Login.aspx)

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)

       Dim conn As SqlConnection
       Dim cmd As SqlCommand
       Dim cmdString As String = "SELECT [Password] FROM [AccessTable] WHERE" & _
          " (([Username] = @Username) AND ([Password] = @Password))"

       conn = New SqlConnection("Data Source=localhost;Initial " & _
          "Catalog=Northwind;Persist Security Info=True;User ID=sa")
       cmd = New SqlCommand(cmdString, conn)

       cmd.Parameters.Add("@Username", SqlDbType.VarChar, 50)
       cmd.Parameters("@Username").Value = TextBox1.Text
```

```
        cmd.Parameters.Add("@Password", SqlDbType.VarChar, 50)
        cmd.Parameters("@Password").Value = TextBox2.Text

        conn.Open()

        Dim myReader As SqlDataReader

        myReader = cmd.ExecuteReader(CommandBehavior.CloseConnection)

        If myReader.Read() Then
           FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, False)
        Else
           Response.Write("Invalid credentials")
        End If

        myReader.Close()
    End Sub
</script>
```

C#

```
<%@ Page Language="C#"%>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        SqlConnection conn;
        SqlCommand cmd;
        string cmdString = "SELECT [Password] FROM [AccessTable] WHERE" +
           " (([Username] = @Username) AND ([Password] = @Password))";

        conn = new SqlConnection("Data Source=localhost;Initial " +
           "Catalog=Northwind;Persist Security Info=True;User ID=sa");
        cmd = new SqlCommand(cmdString, conn);

        cmd.Parameters.Add("@Username", SqlDbType.VarChar, 50);
        cmd.Parameters["@Username"].Value = TextBox1.Text;
        cmd.Parameters.Add("@Password", SqlDbType.VarChar, 50);
        cmd.Parameters["@Password"].Value = TextBox2.Text;

        conn.Open();

        SqlDataReader myReader;

        myReader = cmd.ExecuteReader(CommandBehavior.CloseConnection);

        if (myReader.Read()) {
           FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, false);
        }
        else {
           Response.Write("Invalid credentials");
        }
```

Continued

```
        myReader.Close();
    }
</script>
```

Leave everything else from the previous examples the same, except for the `Login.aspx` page. You can now authenticate usernames and passwords against data stored in SQL Server. In the `Button1_Click` event, a connection is made to SQL Server. (For security reasons, you should store your connection string in the `web.config` file.) Two parameters are passed in — the inputs from `TextBox1` and `TextBox2`. If a result is returned, the `RedirectFromLoginPage()` method is invoked.

Using the Login Control with Forms Authentication

You have seen how to use ASP.NET forms authentication with standard ASP.NET server controls, such as simple TextBox and Button controls. You can also use the ASP.NET server controls — such as the Login server control — with your custom-developed forms-authentication framework instead of using other controls. This really shows the power of ASP.NET — you can combine so many pieces to construct the solution you want.

Listing 21-10 shows a modified `Login.aspx` page using the new Login server control.

Listing 21-10: Using the Login server control on the Login.aspx page

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Login1_Authenticate(ByVal sender As Object, _
      ByVal e As System.Web.UI.WebControls.AuthenticateEventArgs)

        If (Login1.UserName = "BillEvjen" And Login1.Password = "Bubbles") Then
            FormsAuthentication.RedirectFromLoginPage(Login1.UserName, _
               Login1.RememberMeSet)
        Else
            Response.Write("Invalid credentials")
        End If
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Login Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Login ID="Login1" runat="server" OnAuthenticate="Login1_Authenticate">
        </asp:Login>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>
```

```
<script runat="server">
    protected void Login1_Authenticate(object sender, AuthenticateEventArgs e)
    {
        if (Login1.UserName == "BillEvjen" && Login1.Password == "Bubbles") {
            FormsAuthentication.RedirectFromLoginPage(Login1.UserName,
               Login1.RememberMeSet);
        }
        else {
            Response.Write("Invalid credentials");
        }
    }
</script>
```

Because no Button server control is on the page, you use the Login control's `OnAuthenticate` attribute to point to the authentication server-side event — `Login1_Authenticate`. The event takes care of the authorization lookup (although the values are hardcoded in this example). The username text box of the Login control can be accessed via the `Login1.UserName` declaration, and the password can be accessed using `Login1.Password`. The `Login1.RememberMeSet` property is used to specify whether to persist the authentication cookie for the user so that he is remembered on his next visit.

This example is a bit simpler than creating your own login form using TextBox and Button controls. You can give the Login control a predefined look-and-feel that is provided for you. You can also get at the subcontrol properties of the Login control a bit more easily. In the end, it really is up to you as to what methods you employ in your ASP.NET applications.

Looking Closely at the FormsAuthentication Class

As you can tell from the various examples in the forms authentication part of this chapter, a lot of what goes on depends on the `FormsAuthentication` class itself. For this reason, you should learn what that class is all about.

`FormsAuthentication` provides a number of methods and properties that enable you to read and control the authentication cookie as well as other information (such as the return URL of the request). The following table details some of the methods and properties available in the `FormsAuthentictation` class.

Method/Property	**Description**
`Authenticate`	This method is used to authenticate credentials that are stored in a configuration file (such as the `web.config` file).
`Decrypt`	Returns an instance of a valid, encrypted authentication ticket retrieved from an HTTP cookie as an instance of a `FormsAuthenticationTicket` class.
`Encrypt`	Creates a string which contains a valid encrypted authentication ticket that can be used in an HTTP cookie.
`FormsCookieName`	Returns the name of the cookie for the current application.
`FormsCookiePath`	Returns the cookie path (the location of the cookie) for the current application.
`GetAuthCookie`	Provides an authentication cookie for a specified user.

Method/Property	Description
`GetRedirectUrl`	Returns the URL to which the user is redirected after being authorized by the login page.
`HashPasswordForStoring InConfigFile`	Creates a hash of a provided string password. This method takes two parameters — one is the password and the other is the type of hash to perform on the string. Possible hash values include `SHA1` and `MD5`.
`Initialize`	Performs an initialization of the `FormsAuthentication` class by reading the configuration settings in the `web.config` file, as well as getting the cookies and encryption keys used in the given instance of the application.
`RedirectFromLoginPage`	Performs a redirection of the HTTP request back to the original requested page. This should be performed only after the user has been authorized to proceed.
`RenewTicketIfOld`	Conditionally updates the sliding expiration on a `FormsAuthenticationTicket` instance.
`RequireSSL`	Specifies whether the cookie should be transported via SSL only (HTTPS).
`SetAuthCookie`	Creates an authentication ticket and attaches it to a cookie that is contained in the outgoing response.
`SignOut`	Removes the authentication ticket.
`SlidingExpiration`	Provides a Boolean value indicating whether sliding expiration is enabled.

Passport Authentication

Another method for the authentication of your end users is using Microsoft's Passport identity system. Users with a passport account can have a single sign-on solution, meaning that he needs only those credentials to log in to your site and into other Passport-enabled sites and applications on the Internet.

When your application is enabled for Passport authentication, the request is actually redirected to the Microsoft Passport site where the user can enter his credentials. If the authentication is successful, the user is then authorized to proceed, and the request is redirected back to your application.

Very few Internet sites and applications use Microsoft's Passport technologies. In fact, Microsoft has completely de-emphasized Passport in 2005, and most companies interested in global authentication/authorization standards are turning toward the Project Liberty endeavors for a solution (`www.projectliberty.org`).

Authenticating Specific Files and Folders

You may not want to require credentials for each and every page or resource in your application. For instance, you might have a public Internet site with pages anyone can access without credentials, although you might have an administration section as part of your application that may require authentication/authorization measures.

URL authorization enables you to use the `web.config` file to apply the settings you need. Using URL authorization, you can apply any of the authentication measures to only specific files or folders. Listing 21-11 shows an example of locking down a single file.

Listing 21-11: Applying authorization requirements to a single file

```
<configuration>
   <system.web>
      <authentication mode="None" />

      <!-- The rest of your web.config file settings go here -->

   </system.web>

   <location path="AdminPage.aspx">
      <system.web>
         <authentication mode="Windows" />

         <authorization>
            <allow users="ReutersServer\EvjenB" />
            <deny users="*" />
         </authorization>
      </system.web>
   </location>
</configuration>
```

This `web.config` construction keeps the Web application open to the general public while, at the same time, it locks down a single file contained within the application — the `AdminPage.aspx` page. This is accomplished through the `<location>` element. `<location>` takes a single attribute (`path`) to specify the resource defined within the `<system.web>` section of the `web.config` file.

In the example, the `<authentication>` and `<authorization>` elements are used to provide the authentication and authorization details for the `AdminPage.aspx` page. For this page, Windows authentication is applied, and the only user allowed access is `EvjenB` in the `ReutersServer` domain. You can have as many `<location>` sections in your `web.config` file as you want.

Programmatic Authorization

So far, you have seen a lot of authentication examples that simply provide a general authorization to a specific page or folder within the application. Yet, you may want to provide more granular authorization measures for certain items on a page. For instance, you might provide a link to a specific document only for users who have an explicit Windows role. Other users may see something else. You also might want additional commentary or information for specified users, while other users see a condensed version of the information. Whatever your reason, this role-based authorization practice is possible in ASP.NET by working with certain objects.

You can use the `Page` object's `User` property, which provides an instance of the `IPrincipal` object. The `User` property provides a single method and a single property:

- ❑ `Identity`: This property provides an instance of the `System.Security.Principal.IIdentity` object for you to get at specific properties of the authenticated user.

- ❑ `IsInRole`: This method takes a single parameter, a string representation of the system role. It returns a Boolean value that indicates whether the user is in the role specified.

Working with User.Identity

The `User.Identity` property enables you to work with some specific contextual information about the authorized user. Using the property within your ASP.NET applications enables you to make resource-access decisions based on the information the object provides.

With `User.Identity`, you can gain access to the user's name, his authentication type, and whether he is authenticated. The following table details the properties provided through `User.Identity`.

Attribute	Description
`AuthenticationType`	Provides the authentication type of the current user. Example values include `Basic`, `NTLM`, `Forms`, and `Passport`.
`IsAuthenticated`	Returns a Boolean value specifying whether the user has been authenticated.
`Name`	Provides the username of the user as well as the domain of the user (only if he logged on with a Windows account).

For some examples of working with the `User` object, take a look at checking the user's login name. To do this, you use code similar to that shown in Listing 21-12.

Listing 21-12: Getting the username of the logged-in user

VB

```
Dim UserName As String
UserName = User.Identity.Name
```

C#

```
string userName;
userName = User.Identity.Name;
```

Another task you can accomplish with the `User.Identity` object is checking whether the user has been authenticated through your application's authentication methods, as illustrated in Listing 21-13.

Listing 21-13: Checking whether the user is authenticated

VB

```
Dim AuthUser As Boolean
AuthUser = User.Identity.IsAuthenticated
```

C#

```
bool authUser;
authUser = User.Identity.IsAuthenticated;
```

This example provides you with a Boolean value indicating whether the user has been authenticated. You can also use the `IsAuthenticated` method in an `If/Then` statement, as shown in Listing 21-14.

Listing 21-14: Using an If/Then statement that checks authentication

VB

```
If (User.Identity.IsAuthenticated) Then
   ' Do some actions here for authenticated users
Else
   ' Do other actions here for unauthenticated users
End If
```

C#

```
if (User.Identity.IsAuthenticated) {
   // Do some actions here for authenticated users
}
else {
   // Do other actions here for unauthenticated users
}
```

You can also use the `User` object to check the authentication type of the user. This is done with the `AuthenticationType` property illustrated in Listing 21-15.

Listing 21-15: Using the AuthenticationType property

VB

```
Dim AuthType As String
AuthType = User.Identity.AuthenticationType
```

C#

```
string authType;
authType = User.Identity.AuthenticationType;
```

Again, the result is `Basic`, `NTLM`, `Forms`, or `Passport`.

Working with User.IsInRole()

If you are using Windows-based authentication, you can check to make sure that an authenticated user is in a specific Windows role. For example, you might want to show specific information only for users in the `Subscribers` group in the Computer Management Utility. To accomplish that, you can use the `User` object's `IsInRole` method, as shown in Listing 21-16.

Listing 21-16: Checking whether the user is part of a specific role

VB

```
If (User.IsInRole("ReutersServer\Subscribers")) Then
   ' Private information for subscribers
Else
   ' Public information
End If
```

C#

```
if (User.IsInRole("ReutersServer\\Subscribers")) {
   // Private information for subscribers
```

Continued

```
}
else {
   // Public information
}
```

The `IsInRole` method's parameter provides a string value that represents the domain and the group (Windows role). In this case, you specify that any user in the `Subscribers` Windows role from the `ReutersServer` domain is permitted to see some information not available to users who don't belong to that specific role.

Another possibility is to specify some of the built-in groups available to you. Ever since Windows 2000, Windows has included a series of built-in accounts such as Administrator, Guest, PrintOperator, and User. You can access these built-in accounts in a couple of ways. One is to specify the built-in account with the domain directly:

```
User.IsInRole("ReutersServer\Administrator")
```

The other possibility is to use the `BUILTIN` keyword:

```
User.IsInRole("BUILTIN\Administrator")
```

Pulling More Information with WindowsIdentity

So far, in working with the user's identity information, you have used the standard `Identity` object that is part of ASP.NET by default. If you are working with Windows-based authentication, you also have the option of using the `WindowsIdentity` object and other objects. To gain access to these richer objects, create a reference to the `System.Security.Principal` object in your application.

Used in combination with the `Identity` object from the preceding examples, these additional objects make certain tasks even easier. For instance, if you are working with roles, `System.Security.Principal` provides access to the `WindowsBuiltInRole` enumeration.

Listing 21-17 is an example of using the `WindowsBuiltInRole` enumeration.

Listing 21-17: Using the WindowsBuiltInRole enumeration

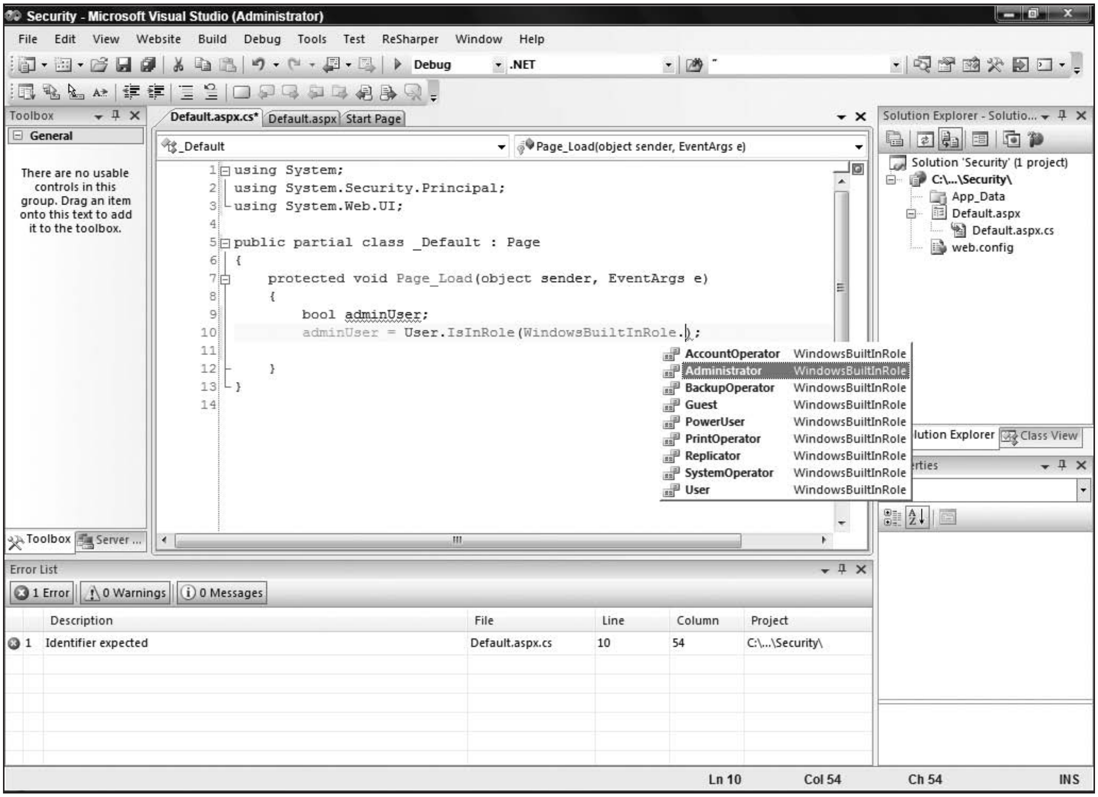
VB

```
Dim AdminUser As Boolean
AdminUser = User.IsInRole(WindowsBuiltInRole.Administrator.ToString())
```

C#

```
bool adminUser;
adminUser = User.IsInRole(WindowsBuiltInRole.Administrator.ToString());
```

Instead of specifying a string value of the domain and the role, you can use the `WindowsBuiltInRole` enumeration to easily access specific roles on the application server. When working with this and other enumerations, you also have IntelliSense (see Figure 21-11) to help you make your selections easily.


Security - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Tools Test ReSharper Window Help
Debug
.NET
Toolbox
General
Default.aspx.cs*
Default.aspx
Start Page
_Default
Page_Load(object sender, EventArgs e)
AccountOperator WindowsBuiltInRole
Administrator WindowsBuiltInRole
BackupOperator WindowsBuiltInRole
Guest WindowsBuiltInRole
PowerUser WindowsBuiltInRole
PrintOperator WindowsBuiltInRole
Replicator WindowsBuiltInRole
SystemOperator WindowsBuiltInRole
User WindowsBuiltInRole
Solution Explorer - Solutio...
Solution 'Security' (1 project)
C:\...\Security\
App_Data
Default.aspx
Default.aspx.cs
web.config
Solution Explorer
Class View
Properties
Toolbox
Server ...
Error List
1 Error
0 Warnings
0 Messages
Ln 10
Col 54
Ch 54
INS

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

```
using System;
using System.Security.Principal;
using System.Web.UI;

public partial class _Default : Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        bool adminUser;
        adminUser = User.IsInRole(WindowsBuiltInRole.);
    }
}
```

	Description	File	Line	Column	Project
1	Identifier expected	Default.aspx.cs	10	54	C:\...\Security\

Figure 21-11

The roles in the `WindowsBuiltInRole` enumeration include the following:

- AccountOperator
- Administrator
- BackupOperator
- Guest
- PowerUser
- PrintOperator
- Replicator
- SystemOperator
- User

Using `System.Security.Principal`, you have access to the `WindowsIdentity` object, which is much richer than working with the default `Identity` object. Listing 21-18 lists some of the additional information you can get through the `WindowsIdentity` object.

Listing 21-18: Using the WindowsIdentity object

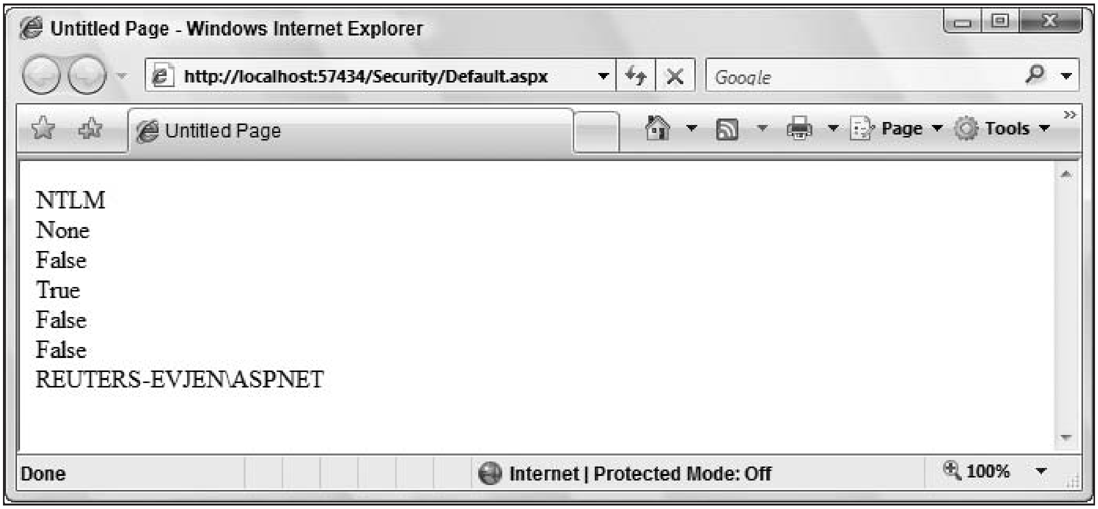
VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Security.Principal" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs)

       Dim AuthUser As WindowsIdentity = WindowsIdentity.GetCurrent()
       Response.Write(AuthUser.AuthenticationType.ToString() & "<br>" & _
          AuthUser.ImpersonationLevel.ToString() & "<br>" & _
          AuthUser.IsAnonymous.ToString() & "<br>" & _
          AuthUser.IsAuthenticated.ToString() & "<br>" & _
          AuthUser.IsGuest.ToString() & "<br>" & _
          AuthUser.IsSystem.ToString() & "<br>" & _
          AuthUser.Name.ToString())
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Security.Principal" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
       WindowsIdentity AuthUser = WindowsIdentity.GetCurrent();
       Response.Write(AuthUser.AuthenticationType.ToString() + "<br>" +
          AuthUser.ImpersonationLevel.ToString() + "<br>" +
          AuthUser.IsAnonymous.ToString() + "<br>" +
          AuthUser.IsAuthenticated.ToString() + "<br>" +
          AuthUser.IsGuest.ToString() + "<br>" +
          AuthUser.IsSystem.ToString() + "<br>" +
          AuthUser.Name.ToString());
    }
</script>
```

In this example, an instance of the `WindowsIdentity` object is created and populated with the current identity of the user accessing the application. Then you have access to a number of properties that are written to the browser using a `Response.Write()` statement. The displayed listing shows information about the current user's credentials, such as if the user is authenticated, anonymous, or running under a guest account or a system account. It also gives you the user's authentication type and login name. A result is shown in Figure 21-12.


Untitled Page - Windows Internet Explorer
http://localhost:57434/Security/Default.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

```
NTLM
None
False
True
False
False
REUTERS-EVJEN\ASPNET
```

Figure 21-12

Identity and Impersonation

By default, ASP.NET runs under an account that has limited privileges. For instance, you may find that although the account can gain access to a network, it cannot be authenticated to any other computer on the network.

The account setting is provided in the `machine.config` file:

```
<processModel
 enable="true"
 userName="machine"
 password="AutoGenerate" />
```

These settings force ASP.NET to run under the system account (ASPNET or Network Service). This is really specified through the `userName` attribute that contains a value of `machine`. The other possible value you can have for this attribute is `system`. Here's what each entails:

- ❑ `machine`: The most secure setting. You should have good reasons to change this value. It's the ideal choice mainly because it forces the ASP.NET account to run under the fewest number of privileges possible.
- ❑ `system`: Forces ASP.NET to run under the local SYSTEM account, which has considerably more privileges to access networking and files.

It is also possible to specify an account of your choosing using the `<processModel>` element in either the `machine.config` or `web.config` files:

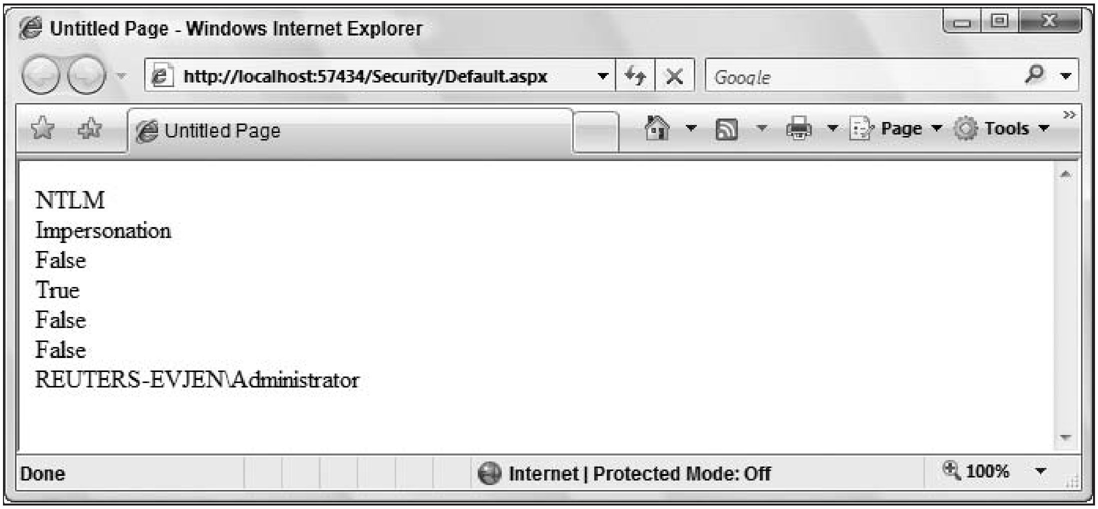
```
<processModel
 enable="true"
 userName="MySpecifiedUser"
 password="MyPassword" />
```

In this example, ASP.NET is run under a specified administrator or user account instead of the default ASPNET or Network Service account. It inherits all the privileges this account offers. You should consider encrypting this section of the file. Encrypting sections of a configuration file are covered in Chapter 32.

You can also change how ASP.NET behaves in whatever account it is specified to run under through the `<identity>` element in the `web.config` file. The `<identity>` element in the `web.config` file allows you to turn on *impersonation*. Impersonation provides ASP.NET with the capability to run as a process using the privileges of another user for a specific session. In more detail, impersonation allows ASP.NET to run under the account of the entity making the request to the application. To turn on this impersonation capability, you use the `impersonate` attribute in the `<identity>` element as shown here:

```
<configuration>
   <system.web>

      <identity impersonate="true" />

   </system.web>
</configuration>
```

By default, the `impersonate` attribute is set to `false`. Setting this property to `true` ensures that ASP.NET runs under the account of the person making the request to the application. If the requestor is an anonymous user, ASP.NET runs under the IUSR_MachineName account. To see this in action, run the example shown in Listing 21-18, but this time with impersonation turned on (`true`). Instead of getting a username of `REUTERS-EVJEN\ASPNET` as the user, you get the name of the user who is requesting the page — `REUTERS-EVJEN\Administrator` in this example — as shown in Figure 21-13.


Untitled Page - Windows Internet Explorer
http://localhost:57434/Security/Default.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

```
NTLM
Impersonation
False
True
False
False
REUTERS-EVJEN\Administrator
```

Figure 21-13

You also have the option of running ASP.NET under a specified account that you declare using the `<identity>` element in the `web.config` file:

```
<identity impersonate="true" userName="MySpecifiedUser" password="MyPassword" />
```

As shown, you can run the ASP.NET process under an account that you specify through the `userName` and `password` attributes. These values are stored as clear text in the `web.config` file.

Look at the root `web.config` file, and you can see that ASP.NET runs under full trust, meaning that it has some pretty high-level capabilities to run and access resources. Here is the setting:

```
<system.web>

    <location allowOverride="true">
        <system.web>
            <securityPolicy>
                <trustLevel name="Full" policyFile="internal"/>
                <trustLevel name="High" policyFile="web_hightrust.config"/>
                <trustLevel name="Medium" policyFile="web_mediumtrust.config"/>
                <trustLevel name="Low" policyFile="web_lowtrust.config"/>
                <trustLevel name="Minimal" policyFile="web_minimaltrust.config"/>
            </securityPolicy>
            <trust level="Full" originUrl=""/>
        </system.web>
    </location>

</system.web>
```

Five possible settings exist for the level of trust that you give ASP.NET — `Full`, `High`, `Medium`, `Low`, and `Minimal`. The level of trust applied is specified through the `<trust>` element's `level` attribute. By default, it is set to `Full`. Each one points to a specific configuration file for the policy in which the level can find its trust level settings. The `Full` setting does not include a policy file because it simply skips all the code access security checks.

Securing Through IIS

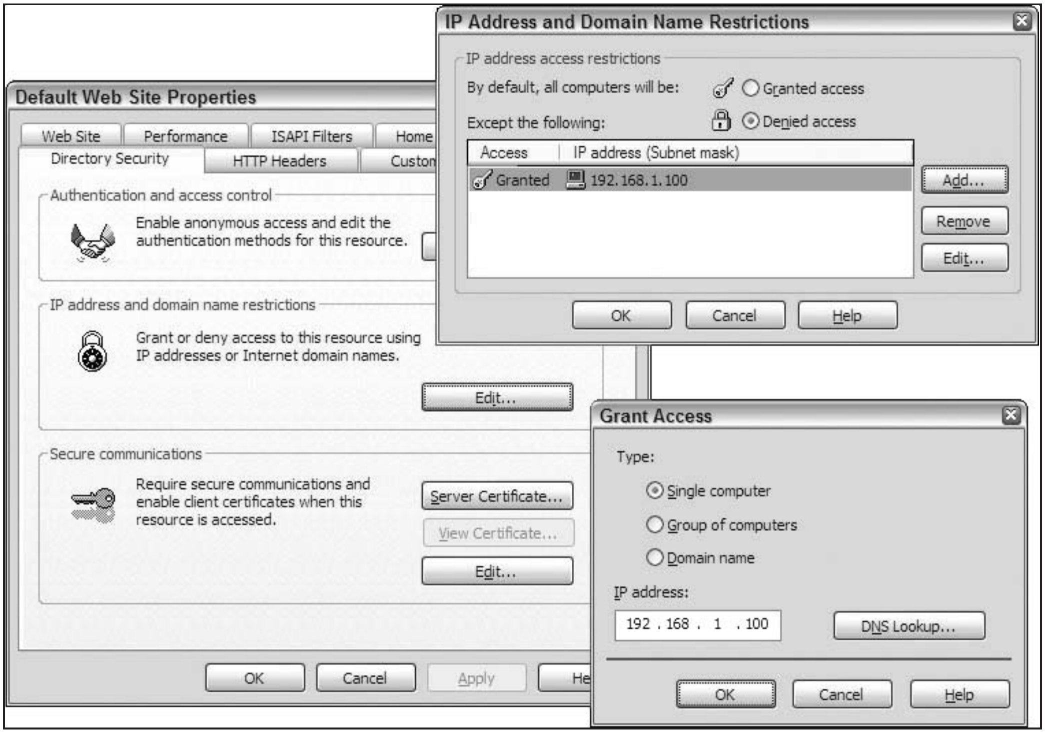
ASP.NET works in conjunction with IIS; not only can you apply security settings directly in ASP.NET (through code or configuration files), but you can also apply additional security measures in IIS itself. IIS enables you to apply access methods you want by working with users and groups (which were discussed earlier in the chapter), working with restricting IP addresses, file extensions, and more. Security through IIS is deserving of a chapter in itself, but the major topics are explored here.

IP Address and Domain Name Restrictions

You can work with the restriction of IP addresses and domain names in Windows Server 2003, Windows 2000 Server, or Windows NT. Through IIS 6.0, you can apply specific restrictions based on a single computer's IP address, a group of computers, or even a specific domain name.

To access this capability, pull up the Internet Information Services (IIS) Manager and right-click on either the Web site you are interested in working with or on the Default Web Site node to simply apply the settings to every Web application on the server. From the menu, choose Properties and select the Directory Security tab.

Click the Edit button in the IP Address and domain name restrictions box and a dialog appears. The resulting dialog enables you to grant or restrict access based on an IP address or domain name. These dialogs are shown in Figure 21-14.



IP Address and Domain Name Restrictions

IP address access restrictions

By default, all computers will be: ○ Granted access ● Denied access

Except the following:

Access	IP address (Subnet mask)
Granted	192.168.1.100

Default Web Site Properties — Directory Security

Authentication and access control: Enable anonymous access and edit the authentication methods for this resource.

IP address and domain name restrictions: Grant or deny access to this resource using IP addresses or Internet domain names.

Secure communications: Require secure communications and enable client certificates when this resource is accessed.

Grant Access

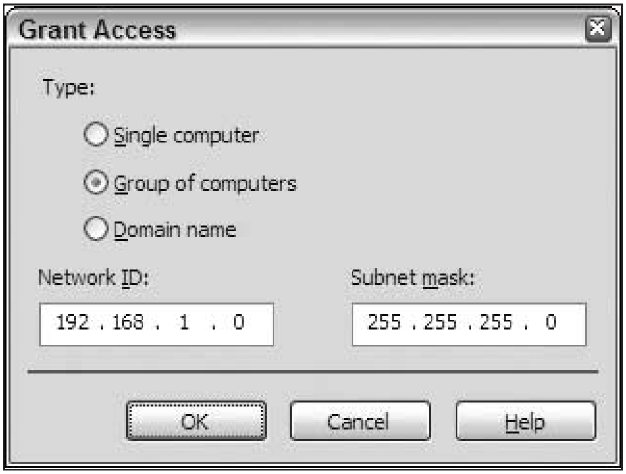
Type:

● Single computer
○ Group of computers
○ Domain name

IP address: 192 . 168 . 1 . 100

Web Site
Performance
ISAPI Filters
Home
Directory Security
HTTP Headers
Custom
Add...
Remove
Edit...
OK
Cancel
Help
Edit...
Server Certificate...
View Certificate...
Edit...
OK
Cancel
Apply
He
DNS Lookup...
OK
Cancel
Help

Figure 21-14



Grant Access

Type:

○ Single computer
● Group of computers
○ Domain name

Network ID: 192 . 168 . 1 . 0

Subnet mask: 255 . 255 . 255 . 0

OK
Cancel
Help

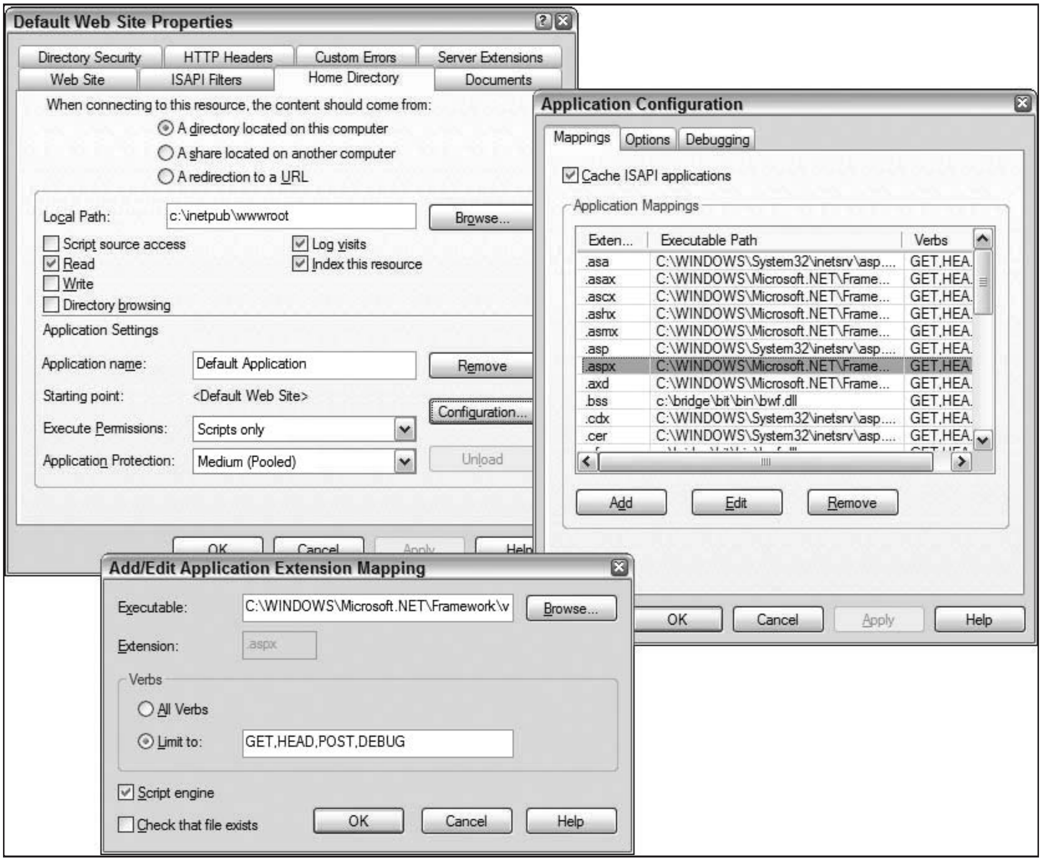
Figure 21-15

Think twice about restricting based on a domain name. It can hinder performance when the reverse DNS lookup is performed on each request to check the domain.

You not only can restrict specific IP addresses and domain names, but you can also restrict everyone and just allow specified entities based on the same items. Although Figure 21-14 shows restricting a specific IP address, you can restrict or grant access to an entire subnet as well. Figure 21-15 shows how to grant access just to the servers on the 192.168.1.0 subnet (defined by a Linksys router).

Working with File Extensions

You can work with many types of files in ASP.NET. These files are defined by their extensions. For example, you know that `.aspx` is a typical ASP.NET page, and `.asmx` is an ASP.NET Web service file extension. These files are actually mapped by IIS to the ASP.NET DLL, `aspnet_isapi.dll`.


Default Web Site Properties
Directory Security
HTTP Headers
Custom Errors
Server Extensions
Web Site
ISAPI Filters
Home Directory
Documents
Browse...
Remove
Configuration...
Unload
OK
Cancel
Apply
Help
Application Configuration
Mappings
Options
Debugging
Application Mappings
Add
Edit
Remove
OK
Cancel
Apply
Help
Add/Edit Application Extension Mapping
Browse...
OK
Cancel
Help

When connecting to this resource, the content should come from:
● A directory located on this computer
○ A share located on another computer
○ A redirection to a URL

Local Path: c:\inetpub\wwwroot
☐ Script source access
☑ Read
☐ Write
☐ Directory browsing
☑ Log visits
☑ Index this resource

Application Settings
Application name: Default Application
Starting point: <Default Web Site>
Execute Permissions: Scripts only
Application Protection: Medium (Pooled)

☑ Cache ISAPI applications

Exten...	Executable Path	Verbs
.asa	C:\WINDOWS\System32\inetsrv\asp....	GET,HEA.
.asax	C:\WINDOWS\Microsoft.NET\Frame...	GET,HEA.
.ascx	C:\WINDOWS\Microsoft.NET\Frame...	GET,HEA.
.ashx	C:\WINDOWS\Microsoft.NET\Frame...	GET,HEA.
.asmx	C:\WINDOWS\Microsoft.NET\Frame...	GET,HEA.
.asp	C:\WINDOWS\System32\inetsrv\asp....	GET,HEA.
.aspx	C:\WINDOWS\Microsoft.NET\Frame...	GET,HEA.
.axd	C:\WINDOWS\Microsoft.NET\Frame...	GET,HEA.
.bss	c:\bridge\bit\bin\bwf.dll	GET,HEA.
.cdx	C:\WINDOWS\System32\inetsrv\asp....	GET,HEA.
.cer	C:\WINDOWS\System32\inetsrv\asp....	GET,HEA.

Executable: C:\WINDOWS\Microsoft.NET\Framework\v
Extension: .aspx
Verbs
○ All Verbs
● Limit to: GET,HEAD,POST,DEBUG
☑ Script engine
☐ Check that file exists

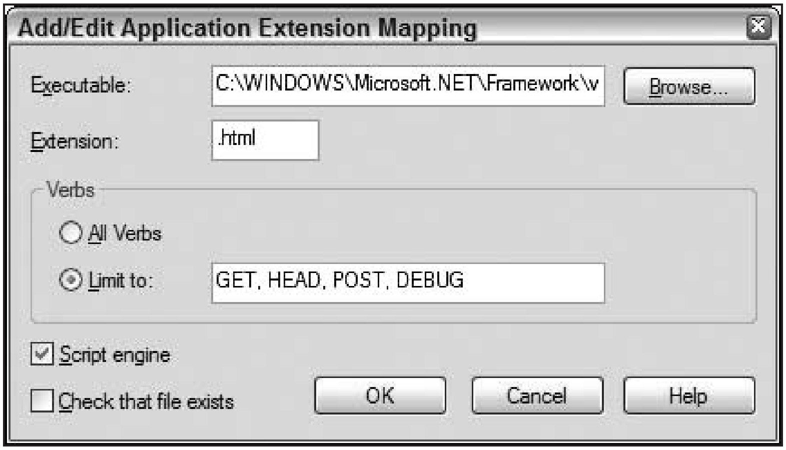
Figure 21-16

To access the dialog in IIS 6.0 that maps the file extensions, pull up the Properties dialog of your Web application in IIS or pull up the Default Web Site Properties. In a specific Web application, you must work from the Directory tab; but if you are working with the Default Web Site Properties dialog, you can instead use the Home Directory tab. From these tabs, click the Configuration button in the Application Settings box. The Application Configuration dialog includes a Mapping tab, where the mappings are configured. Highlight `.aspx` in the list of mappings and click the Edit button. Figure 21-16 shows the result.

In the Executable text box, you can see that all `.aspx` pages map to the `aspnet_isapi.dll` from ASP.NET, and that you can also specify which types of requests are allowed in the application. You can allow either all verbs (for example, `GET` or `POST`) or you can specify which verbs are allowed access to the application.

One important point regarding these mappings is that you do not see `.html`, `.htm`, `.jpg`, or other file extensions such as `.txt` in the list. Your application will not be passing requests for these files to ASP.NET. That might not be a big deal, but in working through the various security examples in this chapter, you might want to have the same type of security measures applied to these files as to `.aspx` pages. If, for instance, you want all `.html` pages to be included in the forms authentication model that you require for your ASP.NET application, you must add `.html` (or whatever file extension you want) to the list. To do so, click the Add button in the Application Configuration dialog.

In the next dialog, you can add the ASP.NET DLL to the Executable text box, and the appropriate file extension and verbs to the list before adding the mapping to your application's mapping table. This example is illustrated in Figure 21-17.



Executable: C:\WINDOWS\Microsoft.NET\Framework\v

Extension: .html

Verbs

○ All Verbs

● Limit to: GET, HEAD, POST, DEBUG

☑ Script engine

☐ Check that file exists

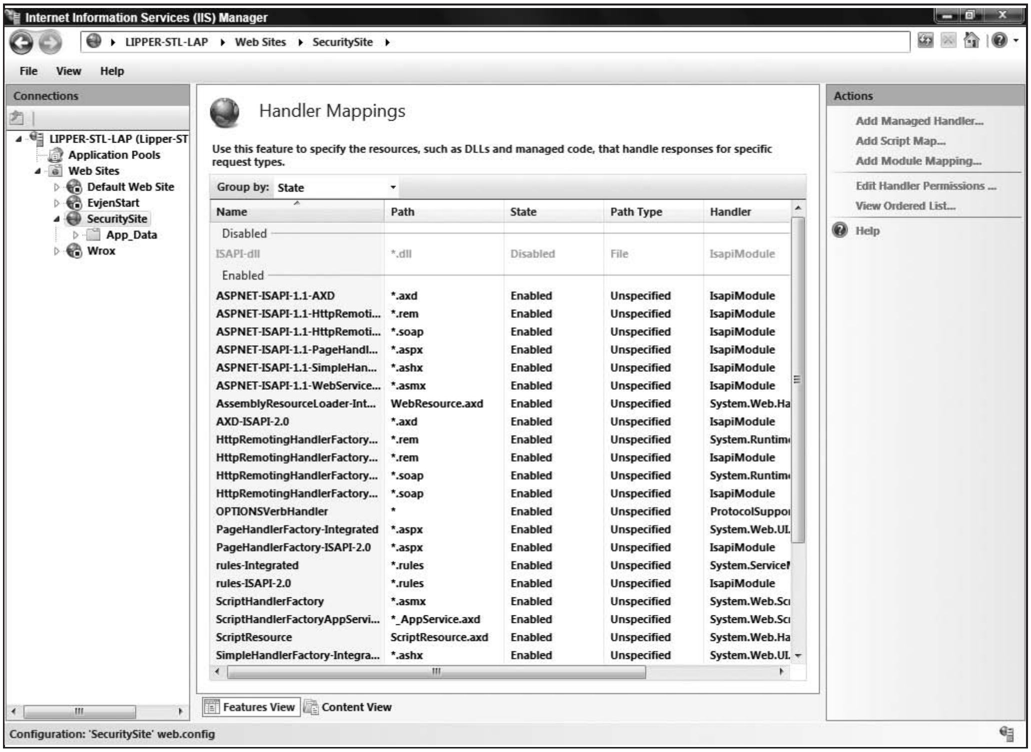
Figure 21-17

When dealing with the security of your site, you have to remember all the files that might not be included in the default mapping list and add the ones you think should fall under the same security structure.

If you are working with Windows Vista, you can get to the same functionality through the IIS Manager. In this tool, select Handler Mappings in the IIS section. You will find a large list of mappings that have already been provided. This is illustrated in Figure 21-18.

By highlighting the `*.aspx` option and pressing the Edit button, you see that this extension is mapped to `%windir%\Microsoft.NET\Framework\v2.0.50727\aspnet_isapi.dll`, as shown in Figure 21-19.

Pressing the Request Restrictions button provides a dialog that enables you to select the verbs allowed (as shown in Figure 21-20).


Internet Information Services (IIS) Manager
LIPPER-STL-LAP ▸ Web Sites ▸ SecuritySite ▸
File View Help
Connections
LIPPER-STL-LAP (Lipper-ST
Application Pools
Web Sites
Default Web Site
EvjenStart
SecuritySite
App_Data
Wrox
Actions
Add Managed Handler...
Add Script Map...
Add Module Mapping...
Edit Handler Permissions ...
View Ordered List...
Help
Features View
Content View

Handler Mappings

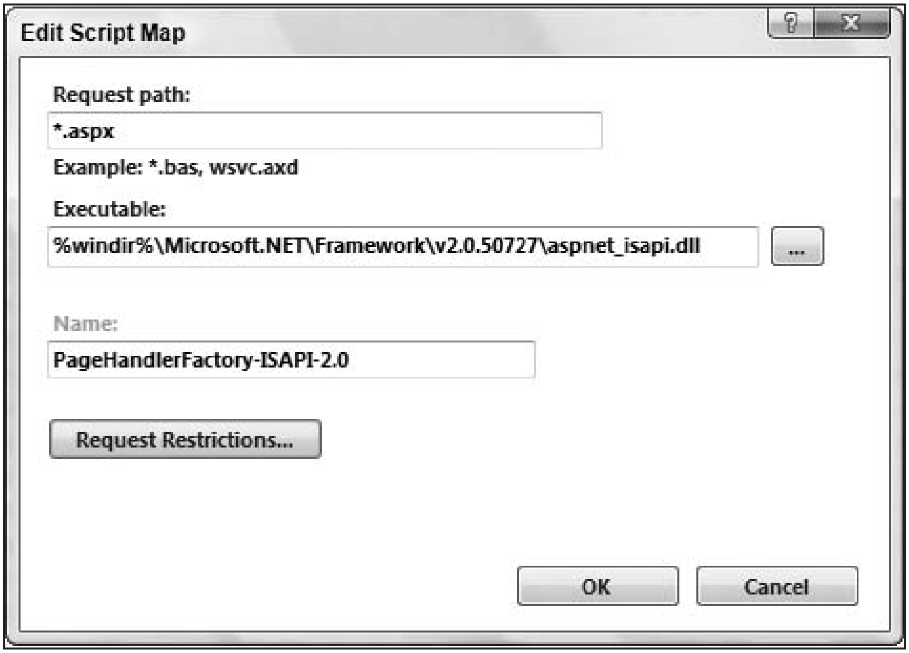
Use this feature to specify the resources, such as DLLs and managed code, that handle responses for specific request types.

Group by: State

Name	Path	State	Path Type	Handler
Disabled				
ISAPI-dll	*.dll	Disabled	File	IsapiModule
Enabled				
ASPNET-ISAPI-1.1-AXD	*.axd	Enabled	Unspecified	IsapiModule
ASPNET-ISAPI-1.1-HttpRemoti...	*.rem	Enabled	Unspecified	IsapiModule
ASPNET-ISAPI-1.1-HttpRemoti...	*.soap	Enabled	Unspecified	IsapiModule
ASPNET-ISAPI-1.1-PageHandl...	*.aspx	Enabled	Unspecified	IsapiModule
ASPNET-ISAPI-1.1-SimpleHan...	*.ashx	Enabled	Unspecified	IsapiModule
ASPNET-ISAPI-1.1-WebService...	*.asmx	Enabled	Unspecified	IsapiModule
AssemblyResourceLoader-Int...	WebResource.axd	Enabled	Unspecified	System.Web.Ha
AXD-ISAPI-2.0	*.axd	Enabled	Unspecified	IsapiModule
HttpRemotingHandlerFactory...	*.rem	Enabled	Unspecified	System.Runtim
HttpRemotingHandlerFactory...	*.rem	Enabled	Unspecified	IsapiModule
HttpRemotingHandlerFactory...	*.soap	Enabled	Unspecified	System.Runtim
HttpRemotingHandlerFactory...	*.soap	Enabled	Unspecified	IsapiModule
OPTIONSVerbHandler	*	Enabled	Unspecified	ProtocolSuppo
PageHandlerFactory-Integrated	*.aspx	Enabled	Unspecified	System.Web.UI.
PageHandlerFactory-ISAPI-2.0	*.aspx	Enabled	Unspecified	IsapiModule
rules-Integrated	*.rules	Enabled	Unspecified	System.Service
rules-ISAPI-2.0	*.rules	Enabled	Unspecified	IsapiModule
ScriptHandlerFactory	*.asmx	Enabled	Unspecified	System.Web.Sc
ScriptHandlerFactoryAppServi...	*_AppService.axd	Enabled	Unspecified	System.Web.Sc
ScriptResource	ScriptResource.axd	Enabled	Unspecified	System.Web.Ha
SimpleHandlerFactory-Integra...	*.ashx	Enabled	Unspecified	System.Web.UI.

Configuration: 'SecuritySite' web.config

Figure 21-18


Edit Script Map
...
Request Restrictions...
OK
Cancel

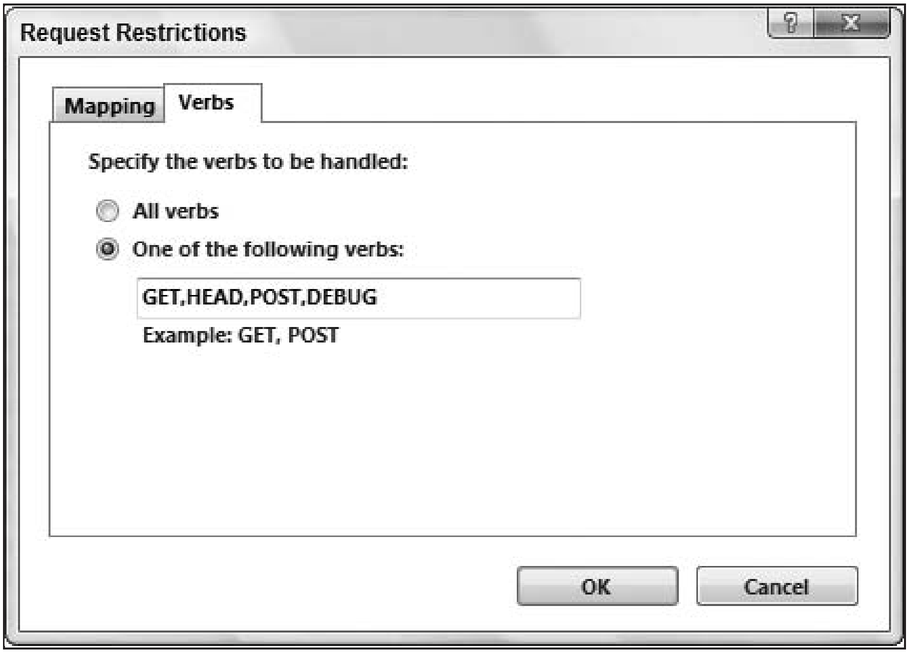
Request path: *.aspx

Example: *.bas, wsvc.axd

Executable: %windir%\Microsoft.NET\Framework\v2.0.50727\aspnet_isapi.dll

Name: PageHandlerFactory-ISAPI-2.0

Figure 21-19



Request Restrictions
Mapping
Verbs
OK
Cancel

Specify the verbs to be handled:

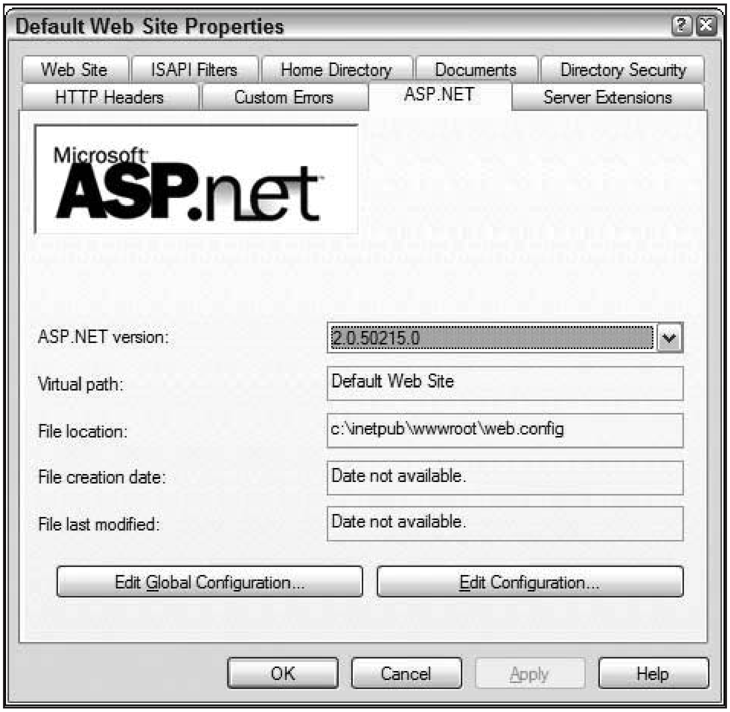
○ All verbs

● One of the following verbs:

GET,HEAD,POST,DEBUG

Example: GET, POST

Figure 21-20



Default Web Site Properties
Web Site
ISAPI Filters
Home Directory
Documents
Directory Security
HTTP Headers
Custom Errors
ASP.NET
Server Extensions
Microsoft ASP.net
Edit Global Configuration...
Edit Configuration...
OK
Cancel
Apply
Help

ASP.NET version: 2.0.50215.0

Virtual path: Default Web Site

File location: c:\inetpub\wwwroot\web.config

File creation date: Date not available.

File last modified: Date not available.

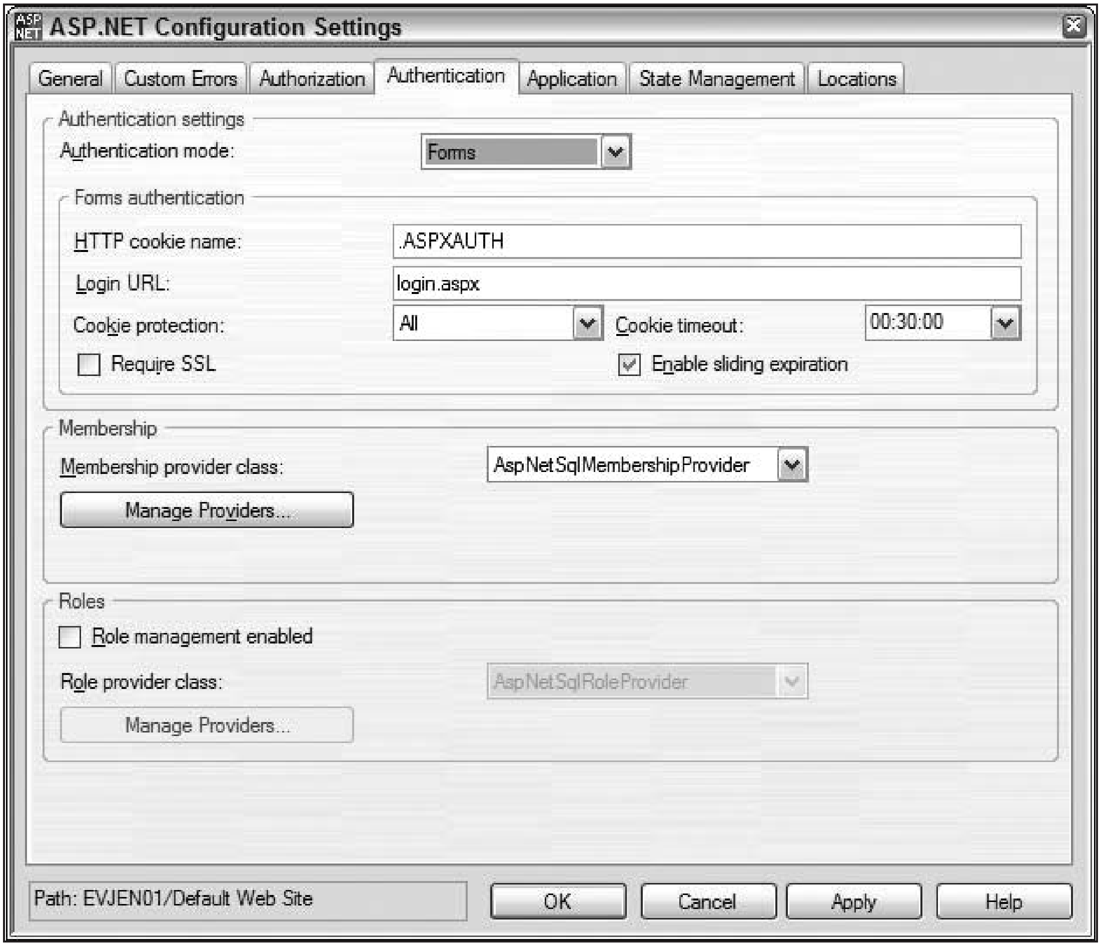
Figure 21-21

Using the ASP.NET MMC Snap-In

The ASP.NET MMC console (covered in more detail in Chapter 34) enables you to edit the `web.config` and `machine.config` files using an easy-to-use GUI instead of having to dig through the text of those files yourself to make the necessary changes. This option is only available in either Windows Server 2003 or Windows XP. Most of the items examined in this book can also be modified and changed using this dialog. The plug-in is available on the ASP.NET tab (see Figure 21-21) of your Web application running under IIS.

When you make the changes directly in the dialog, you are also making the hardcoded changes to the actual configuration files.

Click the Edit Configuration button on the ASP.NET tab, and the ASP.NET Configuration Settings dialog opens. There you can modify how your forms authentication model works in the GUI without going to the application's `web.config` file directly. Figure 21-22 shows an example of working with forms authentication in the GUI.


ASP.NET Configuration Settings
General
Custom Errors
Authorization
Authentication
Application
State Management
Locations
Manage Providers...
Manage Providers...
OK
Cancel
Apply
Help

Authentication settings

Authentication mode: Forms

Forms authentication

HTTP cookie name: .ASPXAUTH

Login URL: login.aspx

Cookie protection: All

Cookie timeout: 00:30:00

☐ Require SSL

☑ Enable sliding expiration

Membership

Membership provider class: AspNetSqlMembershipProvider

Roles

☐ Role management enabled

Role provider class: AspNetSqlRoleProvider

Path: EVJEN01/Default Web Site

Figure 21-22

Using the IIS 7.0 Manager

You will not find the ASP.NET MMC Snap In within Windows Vista. Instead, you will be able to make all the same site modifications through the Internet Information Services (IIS) Manager (as shown in Figure 21-23).

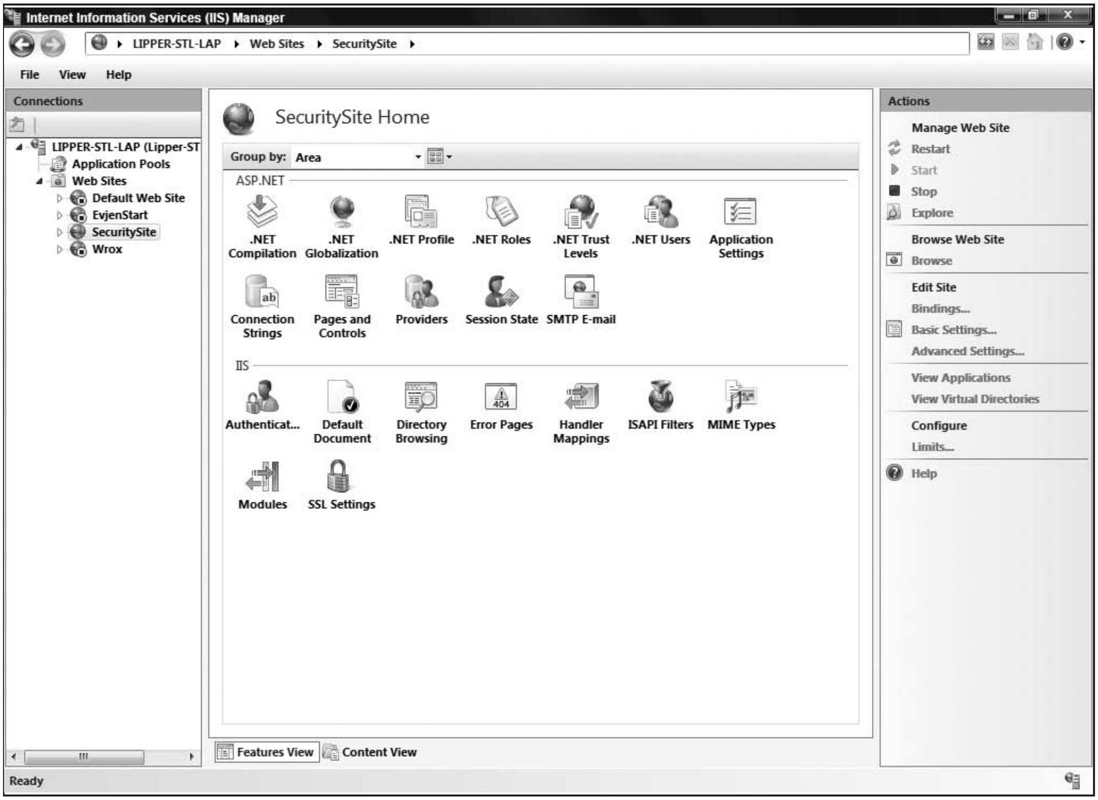

Internet Information Services (IIS) Manager
LIPPER-STL-LAP ▸ Web Sites ▸ SecuritySite ▸
File
View
Help
Connections
LIPPER-STL-LAP (Lipper-ST
Application Pools
Web Sites
Default Web Site
EvjenStart
SecuritySite
Wrox
SecuritySite Home
Group by: Area
ASP.NET
.NET Compilation
.NET Globalization
.NET Profile
.NET Roles
.NET Trust Levels
.NET Users
Application Settings
Connection Strings
Pages and Controls
Providers
Session State
SMTP E-mail
IIS
Authenticat...
Default Document
Directory Browsing
Error Pages
Handler Mappings
ISAPI Filters
MIME Types
Modules
SSL Settings
Features View
Content View
Actions
Manage Web Site
Restart
Start
Stop
Explore
Browse Web Site
Browse
Edit Site
Bindings...
Basic Settings...
Advanced Settings...
View Applications
View Virtual Directories
Configure
Limits...
Help
Ready

Figure 21-23

After making any changes through this dialog, you can select the Apply Changes link in the Actions pane and it will notify you if the changes made have been saved. When successful, the changes made are applied to the site's `web.config` file.

Summary

This chapter covered some of the foundation items of ASP.NET security and showed you how to apply both authentication and authorization to your Web applications. It reviewed some of the various authentication and authorization models at your disposal, such as Basic, Digest, and Windows Integrated Authentication. Other topics included forms-based authentication and how to construct your own forms-based authentication models outside of the ones provided via ASP.NET 3.5 by using the membership and role management capabilities it provides. The chapter also discussed how to use authentication properties within your applications and how to authorize users and groups based on those properties.

22

State Management

Why is state management such a difficult problem that it requires an entire chapter in a book on programming? In the old days (about 15 years ago), using standard client-server architecture meant using a fat client and a fat server. Perhaps your Visual Basic 6 application could talk to a database. The state was held either on the client-side or in the server-side database. Typically, you could count on a client having a little bit of memory and a hard drive of its own to manage state. The most important aspect of traditional client/server design, however, was that the client was *always* connected to the server. It's easy to forget, but HTTP is a stateless protocol. For the most part, a connection is built up and torn down each time a call is made to a remote server. Yes, HTTP 1.1 includes a keep-alive technique that provides optimizations at the TCP level. Even with these optimizations, the server has no way to determine that subsequent connections came from the same client.

Although the Web has the richness of DHTML and Ajax, JavaScript, and HTML 4.0 on the client side, the average high-powered Intel Core Duo with a few gigabytes of RAM is still being used only to render HTML. It's quite ironic that such powerful computers on the client side are still so vastly underutilized when it comes to storing state. Additionally, although many individuals have broadband, it is not universally used. Developers must still respect and pay attention to the dial-up users of the world. When was the last time that your project manager told you that bandwidth was not an issue for your Web application?

The ASP.NET concept of a Session that is maintained over the statelessness of HTTP is not a new one, and it existed before ASP.NET and even before classic ASP. It is, however, a very effective and elegant way to maintain state. There are, however, a number of different choices available to you, of which the ASP.NET session is just one. There have been a few subtle changes between ASP.NET 1.*x* and 2.0/3.5 that will be covered in this chapter. The `Session` object remains as before, but the option to plug in your own session state provider is now available.

What Are Your Choices?

Given a relatively weak client, a stateless protocol such as HTTP, and ASP.NET on the server side, how do you manage state on the Web? Figure 22-1 is a generalized diagram that identifies the primary means available for managing state. The problem is huge, and the solution range is even larger. This chapter assumes that you are not using Java applets or ActiveX controls to manage state. Although these options are certainly valid (although complex) solutions to the state problem, they are beyond the scope of this book.

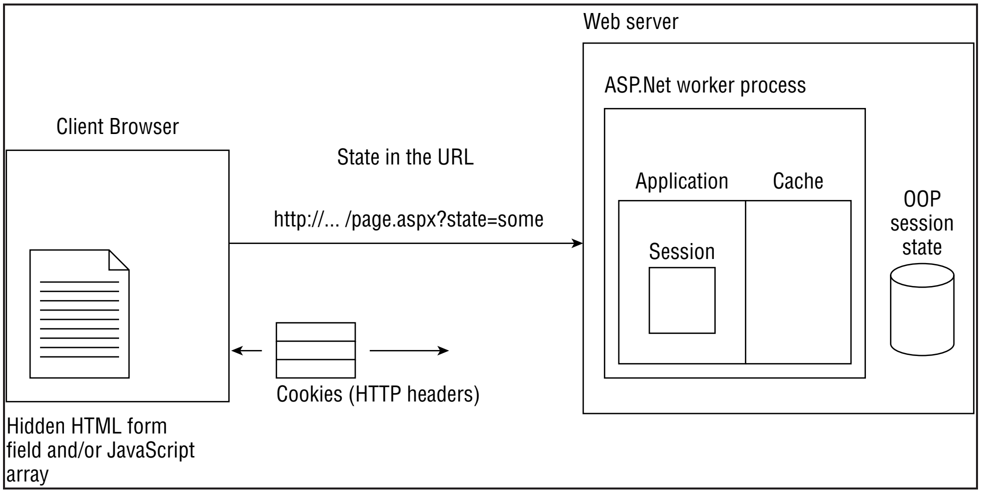


Figure 22-1

If you remember one thing about state management, remember this: There is no right answer. Some answers are more right than others, certainly; but there are many, many ways to manage state. Think about your last project. How many days were spent trying to decide where you should manage state? The trick is to truly understand the pros and cons of each method.

To make an educated decision about a method, you should understand the lifecycle of a request and the opportunities for state management at each point in the process:

1. A Web browser makes an `HTTP GET` request for a page on your server, `http://myserver/myapp/mypage.aspx`. This client Web browser has *never* visited your site before.
2. IIS and your ASP.NET application respond by returning HTML rendered by `mypage.aspx`. Additionally `mypage.aspx` returns a cookie with a unique ID to track this Web browser. Remember that a cookie is actually a slightly abstract concept. The cookie is set by returning a Set-Cookie HTTP Header to the client. The client then promises to return the values of the cookie in every subsequent HTTP call in the HTTP header. The *state* in this example is actually an agreement between the client and server to bounce the cookie back-and-forth on every request in response.
3. The HTML that is returned may contain hidden text boxes such as `<input type="hidden" value="somestate"/>`. These text boxes are similar to cookies because they are passed back to the server if the form on this page is submitted. Cookies are set per domain; hidden form fields are set per page.

4. Upon the next request, the previously set cookies are returned to the server. If this request was the submission of the form as an HTTP POST, all fields in the Form are returned — hidden or otherwise.
5. The unique identifier that was set earlier as a cookie can now be used as a key into any kind of server-side state mechanism. That state might be as simple as an in-memory hashtable, or as complicated as a SQL database.

One of the repeating themes you might notice is the agreement between the client and the server to pass information back and forth. That information can be in the URL, in HTTP headers, or even in the submitted Form as an input field.

On the server side, you have a few options. You'll want to weigh the options based on the amount of memory you have available, the amount of data you want to store, and how often you'll require access to the data.

The following tables express each of the server-side and client-side options and list a few pros and cons for each.

Server-Side Option	Pros	Cons
Application State	Fast. Shared among all users.	State is stored once per server in multiple server configurations.
Cache Object (Application Scope)	Like the Application State but includes expiration via Dependencies (see Chapter 23 on caching).	State is stored once per server in multiple server configurations.
Session State	Three choices: in process, out of process, DB-backed. Can be configured as cookieless.	Can be abused. You pay a serialization cost when objects leave the process. In process requires Web Server affinity. Cookieless configuration makes it easier to hijack.
Database	State can be accessed by any server in a Web farm.	Pay a serialization and persistence cost when objects leave the process. Requires a SQL Server license.

On the client side, every available option costs you in bandwidth. Each option involves passing data back and forth from client to server. Every byte of data you store will be paid for twice: once when it is passed to the server and once when it is passed back.

Client-Side Option	Pros	Cons
Cookie	Simple	Can be rejected by browser. Not appropriate for large amounts of data. Inappropriate for sensitive data. Size cost is paid on *every* HTTP Request and Response.
Hidden Field	Simple for page-scoped data	Not appropriate for large amounts of data. Inappropriate for sensitive data.

Client-Side Option	Pros	Cons
ViewState	Simple for page-scoped data	Encoding of serialized object as binary Base64-encoded data adds approximately 30 percent overhead. Small serialization cost. Has a negative reputation, particularly with DataGrids.
ControlState	Simple for page-scoped control-specific data	Like ViewState, but used for controls that require ViewState even if the developer has turned it off.
QueryString (URL)	Incredibly simple and often convenient if you want your URLs to be modified directly by the end user	Comparatively complex. Can't hold a lot of information. Inappropriate for sensitive data. Easily modified by the end user.

These tables provided you with some of the server-side and client-side options. The improvements to caching in ASP.NET 2.0/3.5 are covered in Chapter 23.

Understanding the Session Object in ASP.NET

In classic ASP, the `Session` object was held in-process (as was everything) to the IIS process. The user received a cookie with a unique key in the form of a GUID. The Session key was an index into a dictionary where object references could be stored.

In all versions of ASP.NET the `Session` object offers an in-process option, but also includes an out-of-process and database-backed option. Additionally, the developer has the option to enable *cookieless* session state where the Session key appears in the URL rather than being sent as a cookie.

Sessions and the Event Model

The `HttpApplication` object raises a series of events during the life of the HTTP protocol request:

- `BeginRequest`: This event fires at the beginning of every request.
- `AuthenticateRequest`: This event is used by the security module and indicates that a request is about to be authenticated. This is where the security module, or you, determines who the user is.
- `AuthorizeRequest`: This event is used by the security module and indicates that a request is about to be authorized. This is where the security module, or you, determines what the user is allowed to do.
- `ResolveRequestCache`: This event is used by the caching module to determine whether this now-authorized request can bypass any additional processing.
- `AcquireRequestState`: This event indicates that all session state associated with this HTTP request is about to be acquired.

Session state is available to you, the developer, *after* the `AcquireRequestState` event fires. The session state key that is unique to each user is retrieved either from a cookie or from the URL.

- ❑ `PreRequestHandlerExecute`: This is the last event you get before the `HttpHandler` class for this request is called.

Your application code, usually in the form of a Page, executes at this point in the process.

- ❑ `PostRequestHandlerExecute`: This is the event that fires just after the `HttpHandler` is called.
- ❑ `ReleaseRequestState`: Indicates that the session state should be stored. Session state is persisted at this point, using whatever Session-state module is configured in `web.config`.
- ❑ `UpdateRequestCache`: All work is complete, and the resulting output is ready to be added to the cache.
- ❑ `EndRequest`: This is the last event called during a request.

You can see from the preceding list that `AcquireRequestState` and `ReleaseRequestState` are two significant events in the life of the `Session` object.

By the time your application code executes, the `Session` object has been populated using the Session key that was present in the cookie, or as you see later, from the URL. If you want to handle some processing at the time the Session begins, rather than handling it in `AcquireRequestState`, you can define an event handler for the `Start` event of a SessionState HttpModule.

```
Sub Session_OnStart()
    'this fires after session state has been acquired by the SessionStateModule.
End Sub
```

The `Session` object includes both `Start` and `End` events that you can hook event handlers to for your own needs. However, the `Session_OnEnd` event is supported only in the In-Process Session State mode. This event will not be raised if you use out-of-process State Server or SQL Server modes. The `Session` ends, but your handlers will never hear about it.

Pre- and post-events occur at almost every point within the life of an HTTP request. Session state can be manipulated at any point after `AcquireRequestState`, including in the `Global.asax` within the `Session_OnStart` event.

The `HttpSessionState` object can be used within any event in a subclass of the `Page` object. Because the pages you create in ASP.NET derive from `System.Web.UI.Page`, you can access Session State as a collection because `System.Web.SessionState.HttpSessionState` implements `ICollection`.

The `Page` has a public property aptly named `Session` that automatically retrieves the `Session` from the current `HttpContext`. Even though it seems as if the `Session` object lives inside the page, it actually lives in the `HttpContext`, and the page's public `Session` property actually retrieves the reference to the session state. This convenience not only makes it more comfortable for the classic ASP programmer, but saves you a little typing as well.

The `Session` object can be referred to within a page in this way:

```
Session["SomeSessionState"] = "Here is some data";
```

or

```
HttpContext.Current.Session["SomeSessionState"] = "Here is some data";
```

The fact that the `Session` object actually lives in the current HTTP context is more than just a piece of trivia. This knowledge enables you to access the `Session` object in contexts other than the page (such as in your own `HttpHandler`).

Configuring Session State Management

All the code within a page refers to the `Session` object using the dictionary-style syntax seen previously, but the `HttpSessionState` object uses a Provider Pattern to extract possible choices for session state storage. You can choose between the included providers by changing the `sessionState` element in `web.config`. ASP.NET ships with the following three storage providers:

- ❑ **In-Process Session State Store:** Stores sessions in the ASP.NET in-memory cache.
- ❑ **Out-Of-Process Session State Store:** Stores sessions in the ASP.NET State Server service `aspnet_state.exe`.
- ❑ **Sql Session State Store:** Stores sessions in Microsoft SQL Server database and is configured with `aspnet_regsql.exe`.

The format of the `web.config` file's `sessionState` element is shown in the following code:

```
<configuration>
   <system.web>
      <sessionState mode="Off|InProc|StateServer|SQLServer|Custom" ../>
   </system.web>
...
```

Begin configuring session state by setting the `mode="InProc"` attribute of the `sessionState` element in the `web.config` of a new Web site. This is the most common configuration for session state within ASP.NET 2.0 and is also the fastest, as you see next.

In-Process Session State

When the configuration is set to `InProc`, session data is stored in the `HttpRuntime's` internal cache in an implementation of `ISessionStateItemCollection` that implements `ICollection`. The session state key is a 120-bit value string that indexes this global dictionary of object references. When session state is in process, objects are stored as live references. This is an incredibly fast mechanism because no serialization occurs, nor do objects leave the process space. Certainly, your objects are not garbage-collected if they exist in the `In-Process Session` object because a reference is still being held.

Additionally, because the objects are stored (held) in memory, they use up memory until that session times out. If a user visits your site and hits one page, he might cause you to store a 40 MB `XmlDocument` in in-process session. If that user never comes back, you are left sitting on that large chunk of memory for the next 20 minutes or so (a configurable value) until the Session ends, even if the user never returns.

InProc Gotchas

Although the InProc Session model is the fastest, the default, and the most common, it does have a significant limitation. If the worker process or application domain recycles, all session state data is lost. Also, the ASP.NET application may restart for a number of reasons, such as the following:

- You've changed the `web.config` or `Global.asax` file or ''touched'' it by changing its modified date.
- You've modified files in the `\bin` or `\App_Code` directory.
- The `processModel` element has been set in the `web.config` or `machine.config` file indicating when the application should restart. Conditions that could generate a restart might be a memory limit or request-queue limit.
- Antivirus software modifies any of the previously mentioned files. This is particularly common with antivirus software that *innoculates* files.

This said, In-Process Session State works great for smaller applications that require only a single Web server, or in situations where IP load balancing is returning each user to the server where his original Session was created.

If a user already has a Session key, but is returned to a different machine than the one on which his session was created, a new Session is created on that new machine using the session ID supplied by the user. Of course, that new Session is empty and unexpected results may occur. However if `regenerateExpiredSessionId` is set to `True` in the `web.config` file, a new Session ID is created and assigned to the user.

Web Gardening

Web gardening is a technique for multiprocessor systems wherein multiple instances of the ASP.NET worker process are started up and assigned with processor affinity. On a larger Web server with as many as four CPUs, you could have anywhere from one to four worker processes hosting ASP.NET. *Processor affinity* means literally that an ASP.NET worker process has an affinity for a particular CPU. It's ''pinned'' to that CPU. This technique is usually enabled only in very large Web farms.

Don't forget that In-Process Session State is just that — in-process. Even if your Web application consists of only a single Web server and all IP traffic is routed to that single server, you have no guarantee that each subsequent request will be served on the same processor. A Web garden must follow many of the same rules that a Web farm follows.

If you're using Web gardening on a multiprocessor system, you must not use In-Process Session State or you lose Sessions. In-Process Session State is appropriate only where there is a 1:1 ratio of applications to application domains.

Storing Data in the Session Object

In the following simple example, in a `Button_Click` event the content of the text box is added to the `Session` object with a specific key. The user then clicks to go to another page within the same application, and the data from the `Session` object is retrieved and presented in the browser.

Note the use of the `<asp:HyperLink>` control. Certainly, that markup could have been hard-coded as HTML, but this small distinction will serve us well later. Additionally, the URL is relative to this site, not absolute. Watch for it to help you later in this chapter.

Listing 22-1 illustrates how simple it is to use the `Session` object. It behaves like any other `IDictionary` collection and allows you to store keys of type `String` associated with any kind of object. The `Retrieve.aspx` file referenced will be added in Listing 22-2.

Listing 22-1: Setting values in session state

ASP.NET

```
<%@ Page Language="C#" CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

ASP.NET — VB.NET

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
    Inherits="_Default" %>
```

ASP.NET

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/
xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Session State</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
        <asp:Button ID="Button1" Runat="server" Text="Store in Session"
        OnClick="Button1_Click" />
        <br />
        <asp:HyperLink ID="HyperLink1" Runat="server"
         NavigateUrl="Retrieve.aspx">Next Page</asp:HyperLink>
    </div>
    </form>
</body>
</html>
```

VB

```
Partial Class _Default
   Inherits System.Web.UI.Page

    Protected Sub Button1_Click(ByVal sender As Object, _
            ByVal e As System.EventArgs)
        Session("mykey") = TextBox1.Text
    End Sub

End Class
```

C#

```
public partial class _Default : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {
        Session["mykey"] = TextBox1.Text;
    }
}
```

The page from Listing 22-1 renders in the browser as shown in Figure 22-2. The `Session` object is accessed as any dictionary indexed by a string key.

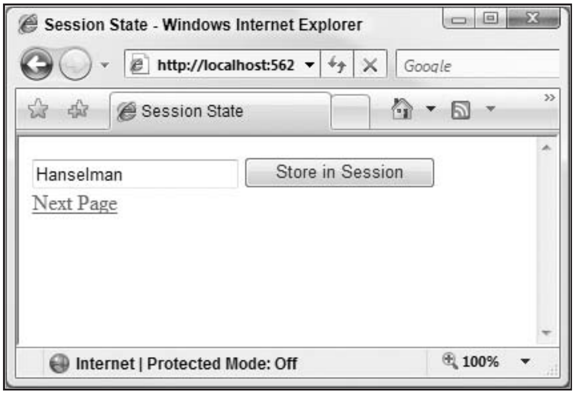


Figure 22-2

More details about the page and the `Session` object can be displayed to the developer if page tracing is enabled. You add this element to your application's `web.config` file inside the `<system.web>` element, as follows:

```
<trace enabled="true" pageOutput="true"/>
```

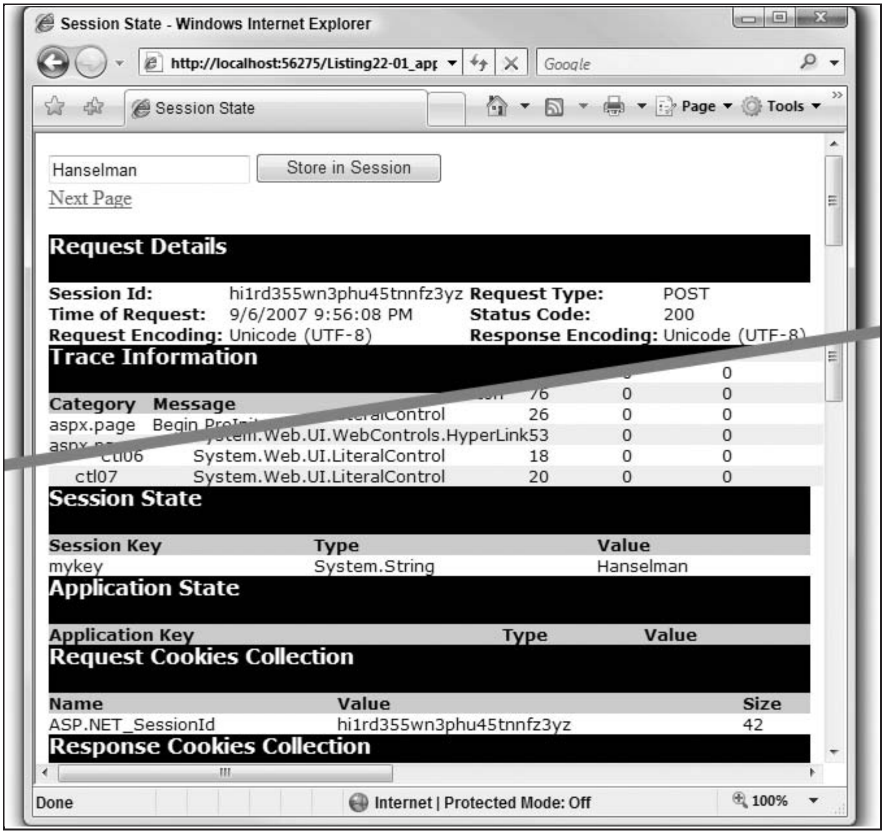
Now tracing is enabled, and the tracing output is sent directly to the page. More details on tracing and debugging are given in Chapter 24. For now, make this change and refresh your browser.

In Figure 22-3, the screenshot is split to show both the top and roughly the middle of the large amount of trace information that is returned when trace is enabled. Session State is very much baked into the fabric of ASP.NET. You can see in the Request Details section of the trace that not only was this page the result of an HTTP POST but the Session ID was as well — elevated to the status of first-class citizen. However, the ASP.NET Session ID lives as a cookie by default, as you can see in the `Cookies` collection at the bottom of the figure.

The default name for that cookie is `ASP.NET_SessionId`, but its name can be configured via the `cookieName` attribute of the `<sessionState>` element in `web.config`. Some large enterprises allow only certain named cookies past their proxies, so you might need to change this value when working on an extranet or a network with a gateway server; but this would be a very rare occurrence. The `cookieName` is changed to use the name ''Foo'' in the following example:

```
<sessionState cookieName="Foo" mode="InProc"></sessionState>
```

The trace output shown in Figure 22-3 includes a section listing the contents of the `Session State` collection. In the figure, you can see that the name `mykey` and the value `Hanselman` are currently stored. Additionally, you see the CLR data type of the stored value; in this case, it's `System.String`.


Session State - Windows Internet Explorer
http://localhost:56275/Listing22-01_apr
Google
Session State
Page
Tools
Store in Session
Next Page
Done
Internet | Protected Mode: Off
100%

Hanselman

Request Details

Session Id:	hi1rd355wn3phu45tnnfz3yz	**Request Type:**	POST
Time of Request:	9/6/2007 9:56:08 PM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message
aspx.page	Begin PreInit

				0
		76	0	0
	LiteralControl	26	0	0
	stem.Web.UI.WebControls.HyperLink	53	0	0
ctl06	System.Web.UI.LiteralControl	18	0	0
ctl07	System.Web.UI.LiteralControl	20	0	0

Session State

Session Key	Type	Value
mykey	System.String	Hanselman

Application State

Application Key	Type	Value

Request Cookies Collection

Name	Value	Size
ASP.NET_SessionId	hi1rd355wn3phu45tnnfz3yz	42

Response Cookies Collection

Figure 22-3

> **The Value column of the trace output comes from a call to the contained object's `ToString()` method. If you store your own objects in the Session, you can override `ToString()` to provide a text-friendly representation of your object that might make the trace results more useful.**

Now add the next page, `retrieve.aspx`, which pulls this value out of the session. Leave the `retrieve.aspx` page as the IDE creates it and add a `Page_Load` event handler, as shown in Listing 22-2.

Listing 22-2: Retrieving values from the session

VB

```
Partial Class Retrieve
   Inherits System.Web.UI.Page
```

```
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles Me.Load

        Dim myValue As String = CType(Session("mykey"), String)
        Response.Write(myValue)
    End Sub
End Class
```

C#

```
public partial class Retrieve : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string myValue = (string)Session["mykey"];
        Response.Write(myValue);
    }
}
```

Because the session contains object references, the resulting object is converted to a string by way of a cast in C# or the `CType` or `CStr` function in VB.

Making Sessions Transparent

It is unfortunate that a cast is usually required to retrieve data from the `Session` object. Combined with the string key used as an index, it makes for a fairly weak contract between the page and the `Session` object. You can create a session helper that is specific to your application to hide these details, or you can add properties to a base `Page` class that presents these objects to your pages in a friendlier way. Because the generic `Session` object is available as a property on `System.Web.UI.Page`, add a new class derived from `Page` that exposes a new property named `MyKey`.

Start by right-clicking your project and selecting Add New Item from the context menu to create a new class. Name it `SmartSessionPage` and click OK. The IDE may tell you that it would like to put this new class in the `/App_Code` folder to make it available to the whole application. Click Yes.

Your new base page is very simple. Via derivation, it does everything that `System.Web.UI.Page` does, plus it has a new property, as shown in Listing 22-3.

Listing 22-3: A more session-aware base page

VB

```
Imports Microsoft.VisualBasic
Imports System
Imports System.Web

Public Class SmartSessionPage
    Inherits System.Web.UI.Page

    Private Const MYSESSIONKEY As String = "mykey"
    Public Property MyKey() As String
        Get
            Return CType(Session(MYSESSIONKEY), String)
        End Get
        Set(ByVal value As String)
            Session(MYSESSIONKEY) = value
        End Set
    End Property
End Class
```

C#

```
using System;
using System.Web;

public class SmartSessionPage : System.Web.UI.Page
{
    private const string MYKEY = "mykey";
    public string MyKey
    {
        get
        {
            return (string)Session[MYKEY];
        }
        set
        {
            Session[MYKEY] = value;
        }
    }
}
```

Now, return to your code from Listing 22-1 and derive your pages from this new base class. To do this, change the base class in the code-beside files to inherit from `SmartSessionPage`. Listing 22-4 shows how the class in the code-behind file derives from the `SmartSessionPage`, which in turn derives from `System.Web.UI.Page`. Listing 22-4 outlines the changes to make to Listing 22-1.

Listing 22-4: Deriving from the new base page

VB — ASPX

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
    Inherits="_Default" %>
```

VB — Default.aspx.vb Code

```
Partial Class _Default
    Inherits SmartSessionPage

   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        ' Session("mykey") = TextBox1.Text
        MyKey = TextBox1.Text
   End Sub
End Class
```

C# — ASPX

```
<%@ Page Language="C#" CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

C# — Default.aspx.cs Code

```
public partial class _Default : SmartSessionPage
```

```
{
    protected void Button1_Click(object sender, EventArgs e)
    {
        //Session["mykey"] = TextBox1.Text;
        MyKey = TextBox1.Text;
    }
}
```

In this code, you change the access to the `Session` object so it uses the new public property. After the changes in Listing 22-3, all derived pages have a public property called `MyKey`. This property can be used without any concern about casting or Session key indexes. Additional specific properties can be added as other objects are included in the Session.

> *Here's an interesting language note: In Listing 22-3 the name of the private string value collides with the public property in VB because they differ only in case. In C#, a private variable named* `MYKEY` *and a public property named* `MyKey` *are both acceptable. Be aware of things like this when creating APIs that will be used with multiple languages. Aim for CLS compliance.*

Advanced Techniques for Optimizing Session Performance

By default, all pages have write access to the `Session`. Because it's possible that more than one page from the same browser client might be requested at the same time (using frames, more than one browser window on the same machine, and so on), a page holds a reader/writer lock on the same `Session` for the duration of the page request. If a page has a writer lock on the same `Session`, all other pages requested in the same `Session` must wait until the first request finishes. To be clear, the `Session` is locked only for that SessionID. These locks don't affect other users with different `Sessions`.

In order to get the best performance out of your pages that use `Session`, ASP.NET allows you declare exactly what your page requires of the `Session` object via the `EnableSessionState @Page` attribute. The options are `True`, `False`, or `ReadOnly`:

- ❑ `EnableSessionState="True"`: The page requires read and write access to the `Session`. The `Session` with that SessionID will be locked during each request.
- ❑ `EnableSessionState="False"`: The page does not require access to the `Session`. If the code uses the `Session` object anyway, an `HttpException` is thrown stopping page execution.
- ❑ `EnableSessionState="ReadOnly"`: The page requires read-only access to the Session. A reader lock is held on the `Session` for each request, but concurrent reads from other pages can occur. The order that locks are requested is essential. As soon as a writer lock is requested, even before a thread is granted access, all subsequent reader lock requests are blocked, regardless of whether a reader lock is currently held or not. While ASP.NET can obviously handle multiple requests, only one request at a time gets write access to a Session.

By modifying the `@Page` direction in `default.aspx` and `retrieve.aspx` to reflect each page's actual need, you affect performance when the site is under load. Add the `EnableSessionState` attribute to the pages, as shown in the following code:

VB — Default.aspx

```
<%@ Page Language="VB" EnableSessionState="True" AutoEventWireup="false"
```

```
    CodeFile="Default.aspx.vb" Inherits="_Default" %>
```

VB — Retrieve.aspx

```
<%@ Page Language="VB" EnableSessionState="ReadOnly" AutoEventWireup="false"
    CodeFile="Retrieve.aspx.vb" Inherits="Retrieve" %>
```

C# — Default.asp

```
<%@ Page Language="C#" EnableSessionState="True"
    CodeFile="Default.aspx.cs" Inherits="_Default"%>
```

C# — Retrieve.aspx

```
<%@ Page Language="C#" EnableSessionState="ReadOnly"
    CodeFile="Retrieve.aspx.cs" Inherits="Retrieve" %>
```

Under the covers, ASP.NET is using marker interfaces from the `System.Web.SessionState` namespace to keep track of each page's needs. When the partial class for `default.aspx` is generated, it implements the `IRequiresSessionState` interface, whereas `Retrieve.aspx` implements `IReadOnlySessionState`. All `HttpRequests` are handled by objects that implement `IHttpHandler`. Pages are handled by a `PageHandlerFactory`. You can find more on HttpHandlers in Chapter 25. Internally, the `SessionStateModule` is executing code similar to the pseudocode that follows:

```
If TypeOf HttpContext.Current.Handler Is IReadOnlySessionState Then
    Return SessionStateStore.GetItem(itemKey)
Else 'If TypeOf HttpContext.Current.Handler Is IRequiresSessionState
    Return SessionStateStore.GetItemExclusive(itemKey)
End If
```

As the programmer, you know things about the intent of your pages at compile time that ASP.NET can't figure out at runtime. By including the `EnableSessionState` attribute in your pages, you allow ASP.NET to operate more efficiently. Remember, ASP.NET always makes the most conservative decision unless you give it more information to act upon.

> ***Performance Tip*: If you're coding a page that doesn't require anything of the Session, by all means, set `EnableSessionState="False"`. This causes ASP.NET to schedule that page ahead of pages that require Session and helps with the overall scalability of your app. Additionally, if your application doesn't use Session at all, set `Mode="Off"` in your `web.config` file to reduce overhead for the entire application.**

Out-of-Process Session State

Out-of-process session state is held in a process called `aspnet_state.exe` that runs as a Windows Service. You can start the ASP.NET state service by using the Services MMC snap-in or by running the following `net` command from an administrative command line:

```
net start aspnet_state
```

By default, the State Service listens on TCP port 42424, but this port can be changed at the registry key for the service, as shown in the following code. The State Service is not started by default.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\
aspnet_state\Parameters\Port
```

Change the `web.config`'s settings from `InProc` to `StateServer`, as shown in the following code. Additionally, you must include the `stateConnectionString` attribute with the IP address and port on which the Session State Service is running. In a Web farm (a group of more than one Web server), you could run the State Service on any single server or on a separate machine entirely. In this example, the State Server is running on the local machine, so the IP address is the localhost IP 127.0.0.1. If you run the State Server on another machine, make sure the appropriate port is open — in this case, TCP port 42424.

```
<configuration>
   <system.web>
      <sessionState mode="StateServer"
          stateConnectionString="tcpip=127.0.0.1:42424"/>
   </system.web>
</configuration>
```

The State Service used is always the most recent one installed with ASP.NET. That means that if you are running ASP.NET 2.0/3.5 and 1.1 on the same machine, all the states stored in `Session` objects for any and all versions of ASP.NET are kept together in a single instance of the ASP.NET State Service.

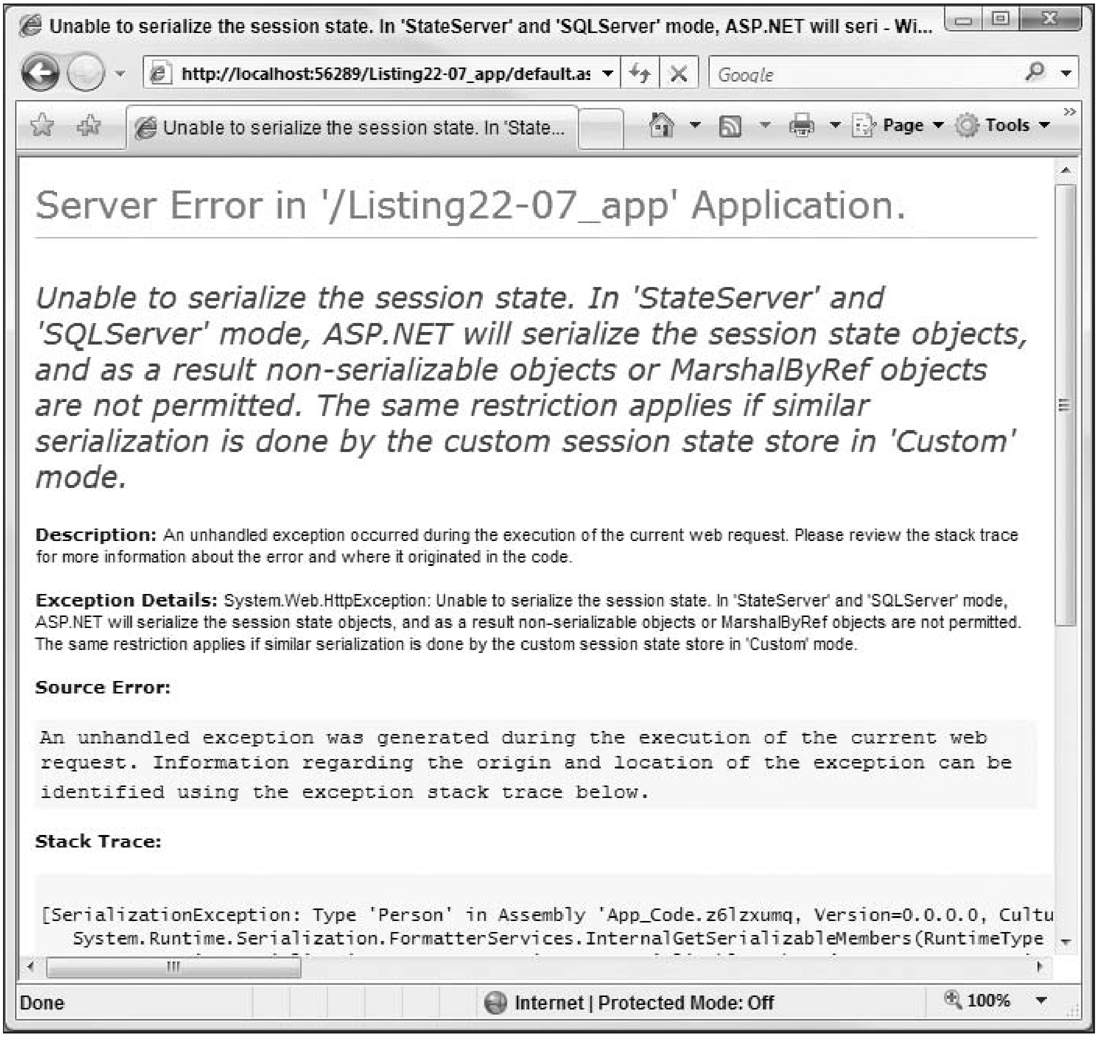
Because your application's code runs in the ASP.NET Worker Process (`aspnet_wp.exe`, or `w3wp.exe`) and the State Service runs in the separate `aspnet_state.exe` process, objects stored in the Session can't be stored as references. Your objects must physically leave the worker process via binary serialization.

> **For a world-class, highly available, and scalable Web site, consider using a Session model other than InProc. Even if you can guarantee via your load-balancing appliance that your Sessions will be *sticky*, you still have application-recycling issues to contend with. The out-of-process state service's data is persisted across application pool recycles but not computer reboots. However, if your state is stored on a different machine entirely, it will survive Web Server recycles and reboots.**

Only classes that have been marked with the `[Serializable]` attribute may be serialized. In the context of the `Session` object, think of the `[Serializable]` attribute as a permission slip for instances of your class to leave the worker process.

Update the `SmartSessionPage` file in your `\App_Code` directory to include a new class called `Person`, as shown in Listing 22-5. Be sure to mark it as `Serializable` or you will see the error shown in Figure 22-4.

As long as you've marked your objects as `[Serializable]`, they'll be allowed out of the ASP.NET process. Notice that the objects in Listing 22-5 are marked `[Serializable]`.



Unable to serialize the session state. In 'StateServer' and 'SQLServer' mode, ASP.NET will seri - Wi...
http://localhost:56289/Listing22-07_app/default.as
Google
Unable to serialize the session state. In 'State...
Page
Tools
Done
Internet | Protected Mode: Off
100%

Server Error in '/Listing22-07_app' Application.

Unable to serialize the session state. In 'StateServer' and 'SQLServer' mode, ASP.NET will serialize the session state objects, and as a result non-serializable objects or MarshalByRef objects are not permitted. The same restriction applies if similar serialization is done by the custom session state store in 'Custom' mode.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Web.HttpException: Unable to serialize the session state. In 'StateServer' and 'SQLServer' mode, ASP.NET will serialize the session state objects, and as a result non-serializable objects or MarshalByRef objects are not permitted. The same restriction applies if similar serialization is done by the custom session state store in 'Custom' mode.

Source Error:

```
An unhandled exception was generated during the execution of the current web
request. Information regarding the origin and location of the exception can be
identified using the exception stack trace below.
```

Stack Trace:

```
[SerializationException: Type 'Person' in Assembly 'App_Code.z6lzxumq, Version=0.0.0.0, Cultu
   System.Runtime.Serialization.FormatterServices.InternalGetSerializableMembers(RuntimeType
```

Figure 22-4

Listing 22-5: A serializable object that can be used in the out-of-process Session

VB

```
<Serializable()> _
Public Class Person
    Public firstName As String
    Public lastName As String

    Public Overrides Function ToString() As String
        Return String.Format("Person Object: {0} {1}", firstName, lastName)
    End Function
End Class
```

C#

```
[Serializable]
public class Person
{
   public string firstName;
   public string lastName;

   public override string ToString()
   {
      return String.Format("Person Object: {0} {1}", firstName, lastName);
   }
}
```

Because you put an instance of the `Person` class from Listing 22-5 into the `Session` object that is currently configured as `StateServer`, you should add a strongly typed property to the base `Page` class from Listing 22-3. In Listing 22-6 you see the strongly typed property added. Note the cast on the property `Get`, and the strongly typed return value indicating that this property deals only with objects of type `Person`.

Listing 22-6: Adding a strongly typed property to SmartSessionPage

VB

```
Public Class SmartSessionPage
    Inherits System.Web.UI.Page

    Private Const MYSESSIONPERSONKEY As String = "myperson"

    Public Property MyPerson() As Person
        Get
            Return CType(Session(MYSESSIONPERSONKEY), Person)
        End Get
        Set(ByVal value As Person)
            Session(MYSESSIONPERSONKEY) = value
        End Set
    End Property

End Class
```

C#

```
public class SmartSessionPage : System.Web.UI.Page
{
    private const string MYPERSON = "myperson";

    public Person MyPerson
    {
        get
        {
            return (Person)Session[MYPERSON];
        }
        set
        {
        Session[MYPERSON] = value;
        }
    }
}
```

Now, add code to create a new `Person`, populate its fields from the text box, and put the instance into the now-out-of-process Session State Service. Then, retrieve the `Person` and write its values out to the browser using the overloaded `ToString()` method from Listing 22-5.

> **Certain classes in the Framework Class Library are not marked as serializable. If you use objects of this type within your own objects, these objects are *not* serializable at all. For example, if you include a DataRow field in a class and add your object to the State Service, you receive a message telling you it ''. . . is not marked as serializable'' because the DataRow includes objects that are not serializable.**

In Listing 22-7, the value of the `TextBox` is split into a string array and the first two strings are put into a `Person` instance. For example, if you entered `"Scott Hanselman"` as a value, `"Scott"` is put into `Person.firstName` and `"Hanselman"` is put into `Person.lastName`. The values you enter should appear when they are retrieved later in `Retrieve.aspx` and written out to the browser with the overloaded `ToString` method.

Listing 22-7: Setting and retrieving objects from the Session using State Service and a base page

VB — Default.aspx.vb

```
Partial Class _Default
    Inherits SmartSessionPage

   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim names As String()
        names = TextBox1.Text.Split(" "c) ' " "c creates a char
        Dim p As New Person()
        p.firstName = names(0)
        p.lastName = names(1)
        Session("myperson") = p
   End Sub
End Class
```

VB — Retrieve.aspx.vb

```
Partial Class Retrieve
        Inherits SmartSessionPage
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
            Handles Me.Load
        Dim p As Person = MyPerson
        Response.Write(p) ' ToString will be called!
    End Sub
End Class
```

C# — Default.aspx.cs

```
public partial class _Default : SmartSessionPage
{
    protected void Button1_Click(object sender, EventArgs e)
```

```
        {
            string[] names = TextBox1.Text.Split(' ');
            Person p = new Person();
            p.firstName = names[0];
            p.lastName = names[1];

            Session["myperson"] = p;
        }
    }
```

C# — Retrieve.aspx.cs

```
public partial class Retrieve : SmartSessionPage
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Person p = MyPerson;
        Response.Write(p); //ToString will be called!
    }
}
```

Now, launch the browser, enter your name (or `"Scott Hanselman"` if you like), click the button to store it in the `Session`, and then visit `Retrieve.aspx` via the hyperlink. You see the result of the `ToString()` method via `Response.Write`, as shown in Figure 22-5.

The completed code and techniques shown in Listing 22-7 illustrate a number of best practices for session management:

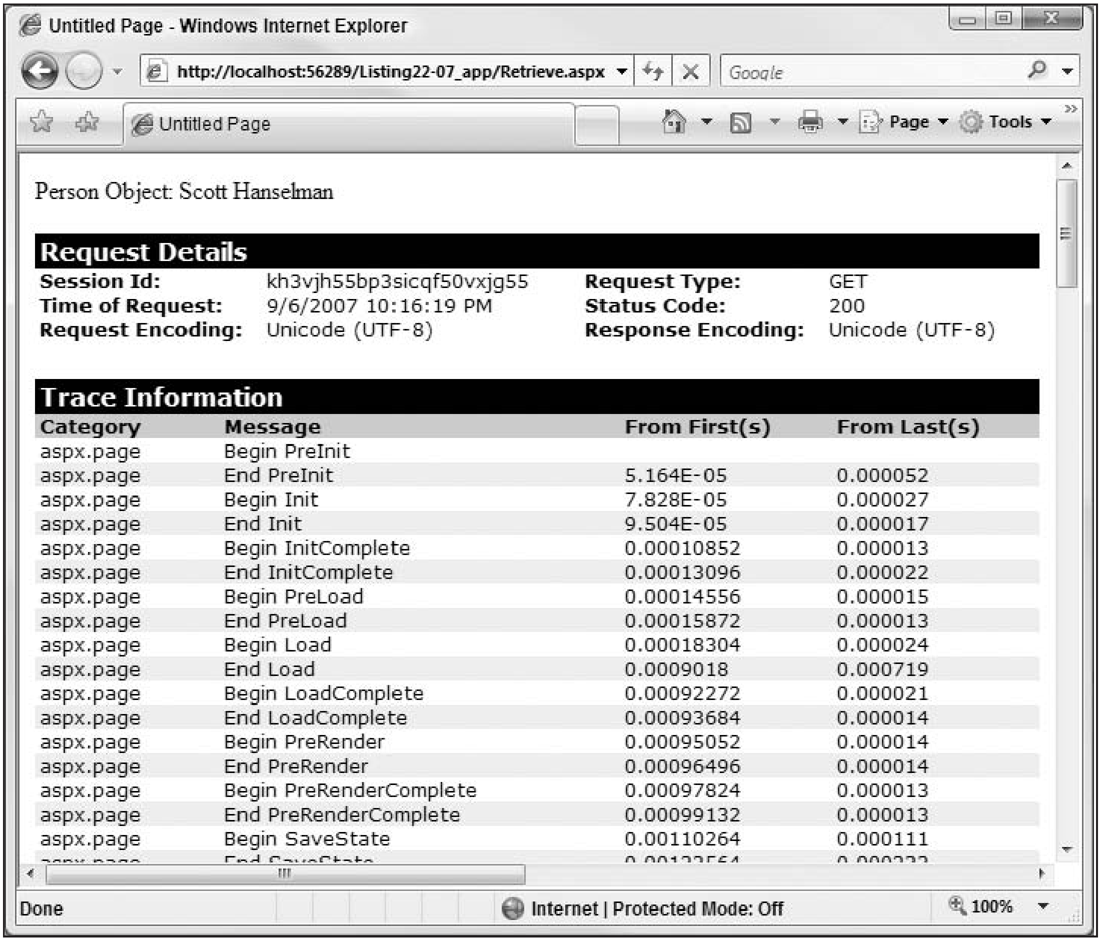
- ❑ Mark your objects as `Serializable` if you might ever use non-In-Proc session state.
- ❑ Even better, do all your development with a local session state server. This forces you to discover non-serializable objects early, gives you a sense of the performance and memory usages of `aspnet_state.exe`, and allows you to choose from any of the session options at deployment time.
- ❑ Use a base `Page` class or helper object with strongly typed properties to simplify your code. It enables you to hide the casts made to session keys otherwise referenced throughout your code.

These best practices apply to all state storage methods, including SQL session state.

SQL-Backed Session State

ASP.NET sessions can also be stored in a SQL Server database. InProc offers speed, StateServer offers a resilience/speed balance, and storing sessions in SQL Server offers resilience that can serve sessions to a large Web farm that persists across IIS restarts, if necessary.

SQL-backed session state is configured with `aspnet_regsql.exe`. This tool adds and removes support for a number of ASP.NET features such as cache dependency (see Chapter 23) and personalization/membership (see Chapters 17 and 18) as well as session support. When you run `aspnet_regsql.exe` from the command line without any options, surprisingly, it pops up a GUI as shown in Figure 22-6. This utility is located in the .NET Framework's installed directory, usually `c:\windows\microsoft.net\framework\2.0`. Note that the Framework 3.5 includes additional libraries, but this wizard is still in the 2.0 directory.


Untitled Page - Windows Internet Explorer
http://localhost:56289/Listing22-07_app/Retrieve.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

Person Object: Scott Hanselman

Request Details

Session Id:	kh3vjh55bp3sicqf50vxjg55	**Request Type:**	GET
Time of Request:	9/6/2007 10:16:19 PM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

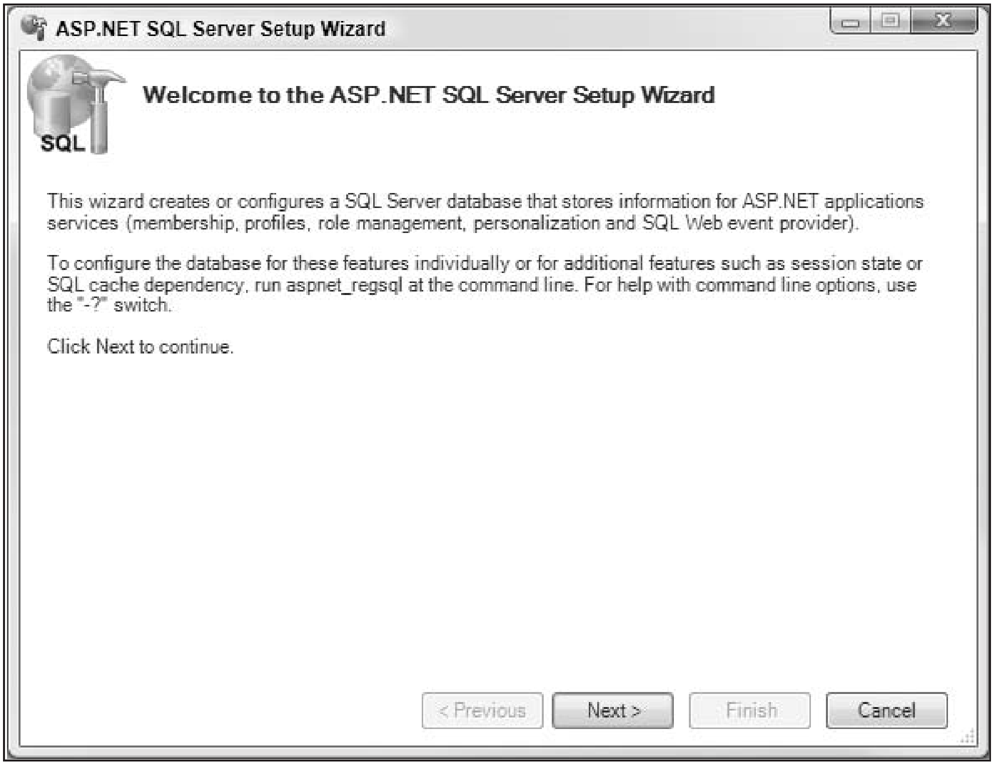
Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	5.164E-05	0.000052
aspx.page	Begin Init	7.828E-05	0.000027
aspx.page	End Init	9.504E-05	0.000017
aspx.page	Begin InitComplete	0.00010852	0.000013
aspx.page	End InitComplete	0.00013096	0.000022
aspx.page	Begin PreLoad	0.00014556	0.000015
aspx.page	End PreLoad	0.00015872	0.000013
aspx.page	Begin Load	0.00018304	0.000024
aspx.page	End Load	0.0009018	0.000719
aspx.page	Begin LoadComplete	0.00092272	0.000021
aspx.page	End LoadComplete	0.00093684	0.000014
aspx.page	Begin PreRender	0.00095052	0.000014
aspx.page	End PreRender	0.00096496	0.000014
aspx.page	Begin PreRenderComplete	0.00097824	0.000013
aspx.page	End PreRenderComplete	0.00099132	0.000013
aspx.page	Begin SaveState	0.00110264	0.000111

Figure 22-5

The text of the dialog shown in Figure 22-6 contains instructions to run `aspnet_regsql` from the command line with a `"-?"` switch. You have a huge number of options, so you'll want to pipe it through in a form like `aspnet_regsql -? | more`. You see the session-state–specific options shown here:

```
                          -- SESSION STATE OPTIONS --

   -ssadd                 Add support for SQLServer mode session state.

   -ssremove              Remove support for SQLServer mode session state.

   -sstype t|p|c          Type of session state support:

                          t: temporary. Session state data is stored in the
                          "tempdb" database. Stored procedures for managing
                          session are installed in the "ASPState" database.
                          Data is not persisted if you restart SQL. (Default)
```

```
                            p: persisted. Both session state data and the stored
                            procedures are stored in the "ASPState" database.

                            c: custom. Both session state data and the stored
                            procedures are stored in a custom database. The
                            database name must be specified.

    -d <database>            The name of the custom database to use if -sstype is
                            "c".
```



Welcome to the ASP.NET SQL Server Setup Wizard

This wizard creates or configures a SQL Server database that stores information for ASP.NET applications services (membership, profiles, role management, personalization and SQL Web event provider).

To configure the database for these features individually or for additional features such as session state or SQL cache dependency, run aspnet_regsql at the command line. For help with command line options, use the "-?" switch.

Click Next to continue.

Figure 22-6

Three options exist for session state support: `t`, `p`, and `c`. The most significant difference is that the `-sstype t` option does not persist session state data across SQL Server restarts, whereas the `-sstype p` option does. Alternatively, you can specify a custom database with the `-c` option and give the database name with `-d database`.

The following command-line example configures your system for SQL session support with the SQL Server on localhost with an `sa` password of *wrox* and a persistent store in the ASPState database. (Certainly, you know not to deploy your system using `sa` and a weak password, but this simplifies the example. Ideally you'd use Windows Integration Authentication and give the Worker Process Identity access to the ASPState database.) If you're using SQL Express, replace ''localhost'' with ''.\SQLEXPRESS''. If you aren't using Windows Authentication, you may need to explicitly enable the sa account from the Management Studio, run this tool, and then disable the sa account for security reasons.

```
C:\ >aspnet_regsql -S localhost -U sa -P wrox -ssadd -sstype p
Start adding session state.
...........
Finished.
```

Next, open up Enterprise Manager and look at the newly created database. Two tables are created — ASPStateTempApplications and ASPStateTempSessions — as well as a series of stored procedures to support moving the session back and forth from SQL to memory.

If your SQL Server has its security locked down tight, you might get an Error 15501 after executing `aspnet_regsql.exe` that says ''An error occurred during the execution of the SQL file 'InstallSqlState.sql'.'' The SQL error number is 15501 and the SqlException message is:

```
This module has been marked OFF. Turn on 'Agent XPs' in order to be able to access the module. If
the job does not exist, an error from msdb.dbo.sp_delete_job is expected.
```

This is a rather obscure message, but `aspnet_regsql.exe` is trying to tell you that the extended stored procedures it needs to enable session state are not enabled for security reasons. You'll need to allow them explicitly. To do so, execute the following commands within the SQL Server 2005 Query Analyzer or the SQL Server 2005 Express Manager:

```
USE master
EXECUTE sp_configure 'show advanced options', 1
RECONFIGURE WITH OVERRIDE
GO
EXECUTE sp_configure 'Agent XPs', 1
RECONFIGURE WITH OVERRIDE
GO
EXECUTE sp_configure 'show advanced options', 0
RECONFIGURE WITH OVERRIDE
GO
```

Now, change the `web.config` `<sessionState>` element to use SQL Server, as well as the new connection string:

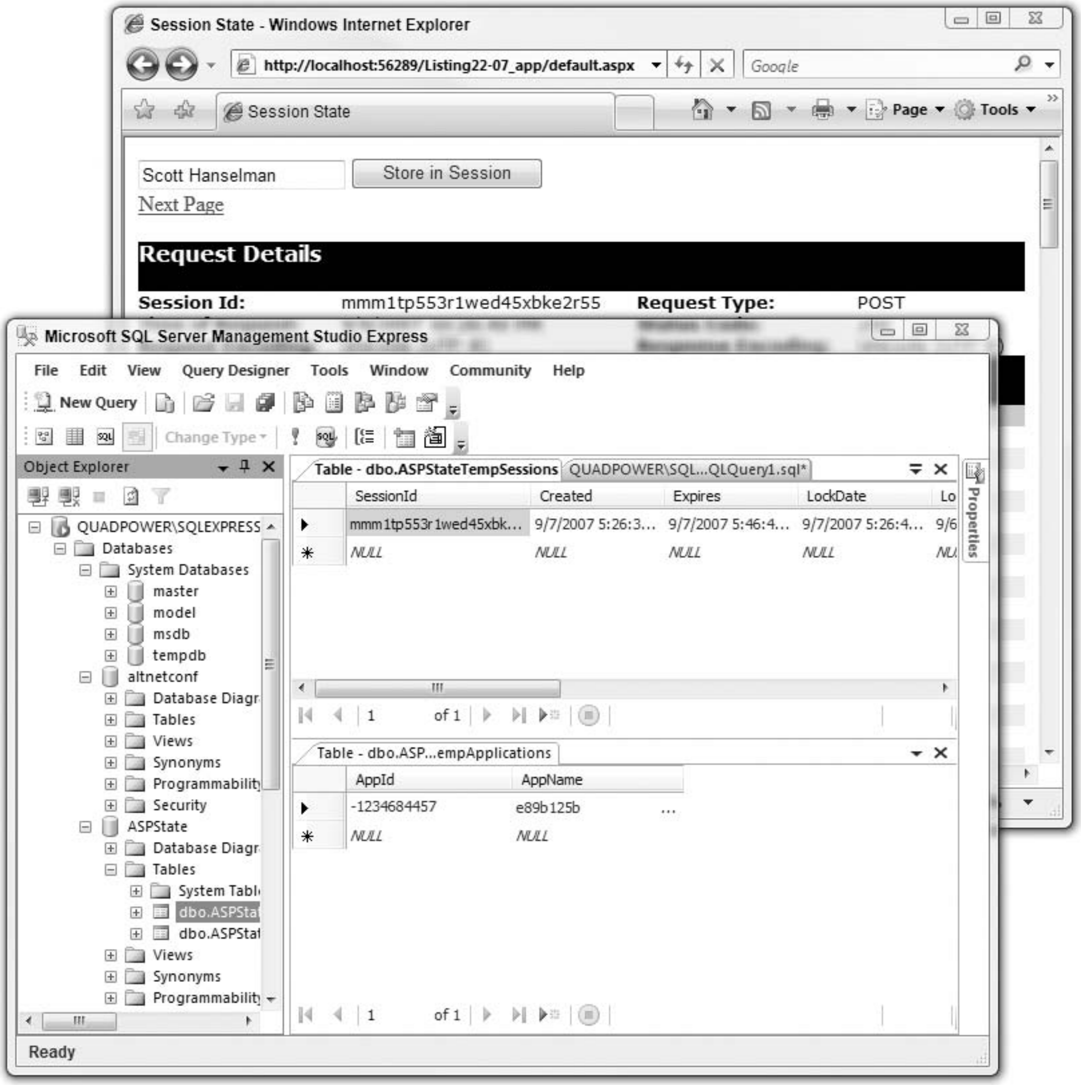
```
<sessionState mode="SQLServer" sqlConnectionString="data source=127.0.0.1;user
id=sa;password=Wrox"/>
```

The session code shown in Listing 22-7 continues to work as before. However, if you open up the ASPStateTempSessions table, you see the serialized objects. Notice in Figure 22-7 that the Session ID from the trace appears as a primary key in a row in the ASPStateTempSessions table.

Figure 22-7 shows the `SessionId` as seen in the Request Details of ASP.NET tracing. That `SessionId` appears in the `SessionId` column of the `ASPStateTempSessions` table in the `ASPState` database just created. Notice also the `ASPStateTempApplications` table that keeps track of each IIS application that may be using the same database to manage sessions.

If you want to use your own database to store session state, you specify the database name with the `-d <database>` switch of `aspnet_regsql.exe` and include the `allowCustomSqlDatabase="true"` attribute and the name of the database in the connection string:

```
<sessionState allowCustomSqlDatabase="true" mode="SQLServer"
sqlConnectionString="data source=127.0.0.1; database=MyCustomASPStateDatabase;"/>
```


Session State - Windows Internet Explorer
http://localhost:56289/Listing22-07_app/default.aspx
Google
Session State
Page
Tools
Store in Session
Next Page
Microsoft SQL Server Management Studio Express
File
Edit
View
Query Designer
Tools
Window
Community
Help
New Query
Change Type
Object Explorer
QUADPOWER\SQLEXPRESS
Databases
System Databases
master
model
msdb
tempdb
altnetconf
Database Diagr
Tables
Views
Synonyms
Programmability
Security
ASPState
Database Diagr
Tables
System Tabl
dbo.ASPStat
dbo.ASPStat
Views
Synonyms
Programmability
Table - dbo.ASPStateTempSessions
QUADPOWER\SQL...QLQuery1.sql*
Table - dbo.ASP...empApplications
1 of 1
Properties
Ready

Scott Hanselman

Request Details

Session Id: mmm1tp553r1wed45xbke2r55 **Request Type:** POST

SessionId	Created	Expires	LockDate	Lo
mmm1tp553r1wed45xbk...	9/7/2007 5:26:3...	9/7/2007 5:46:4...	9/7/2007 5:26:4...	9/6
NULL	NULL	NULL	NULL	NU

AppId	AppName
-1234684457	e89b125b
NULL	NULL

Figure 22-7

The user ID and password can be included in the connection string; or Windows Integrated Security can be used if the ASP.NET Worker Process's identity is configured with access in SQL Server.

Extending Session State with Other Providers

ASP.NET Session State is built on a new, extensible, provider-based storage model. You can implement custom providers that store session data in other storage mechanisms simply by deriving from `SessionStateStoreProviderBase`. This extensibility feature also allows you to generate session IDs via your own algorithms by implementing `ISessionIDManager`.

You start by creating a class that inherits from `SessionStateStoreProviderBase`. The session module will call methods on any session provider as long as it derives from `SessionStateStoreProviderBase`. Register your custom provider in your application's `web.config`, as in the following example:

```
<sessionState mode ="Custom" customProvider ="WroxProvider">
    <providers >
        <add name ="WroxProvider" type ="Wrox.WroxStore, WroxSessionSupplier"/>
    </providers>
</sessionState>
```

ASP.NET initializes the `SessionStateModule`, and these methods are called on any custom implementation:

- `Initialize`: This method is inherited ultimately from `System.Configuration.Provider.ProviderBase` and is called immediately after the constructor. With this method, you set your provider name and call up to the base implementation of `Initialize`.
- `SetItemExpireCallback`: With this method, you can register any methods to be called when a session item expires.
- `InitializeRequest`: This method is called by the `SessionStateModule` for each request. This is an early opportunity to get ready for any requests for data that are coming.
- `CreateNewStoreData`: With this method, you create a new instance of `SessionStateStoreData`, the data structure that holds session items, the session timeout values, and any static items.

When a session item is requested, ASP.NET calls your implementation to retrieve it. Implement the following methods to retrieve items:

- `GetItemExclusive`: This method enables you to get `SessionStateStoreData` from your chosen store. You may have created an Oracle provider, stored data in XML, or stored data elsewhere.
- `GetItem`: This is your opportunity to retrieve it as you did in `GetItemExclusive` except without exclusive locking. You may or may not care, depending on what backing store you've chosen.

When it's time to store an item, the following method is called:

- SetAndReleaseItemExculsive: Here you should save the `SessionStateStoreData` object to your custom store.

A number of third-party session state providers are available — both open source and for sale.

ScaleOut Software released the first third-party ASP.NET State Provider in the form of their StateServer product. It fills a niche between the ASP.NET included singleton StateServer and the SQL Server Database State Provider. ScaleOut Software's StateServer is an out-of-process service that runs on each machine in the Web farm and ensures that session state is stored in a transparent and distributed manner among machines in the farm. You can learn more about StateServer and their ASP.NET 2.0 Session Provider at `www.scaleoutsoftware.com/`.

Another commercial product is StrangeLoop Network's AppScaler. It includes an ASP.NET Session Provider that plugs into their network acceleration appliance. Their product also optimizes ViewState among other things and won a "Best of TechEd 2007" award. Details are at `www.strangeloopnetworks.com/`.

The derivation-based provider module for things such as session state will no doubt continue to create a rich ecosystem of enthusiasts who will help push the functionality to new places Microsoft did not expect.

Cookieless Session State

In the previous example, the ASP.NET Session State ID was stored in a cookie. Some devices don't support cookies, or a user may have turned off cookie support in his browser. Cookies are convenient because the values are passed back and forth with every request and response. That means every `HttpRequest` contains cookie values, and every `HttpResponse` contains cookie values. What is the only other thing that is passed back and forth with every `Request` and `Response`? The URL.

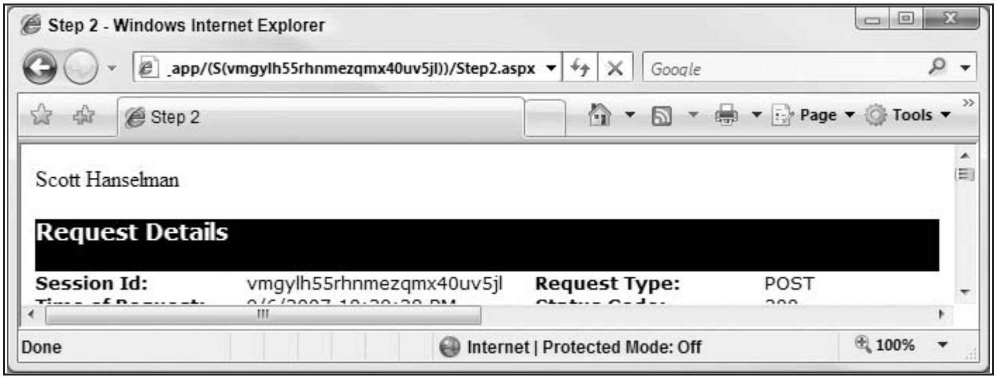
If you include the `cookieless="UseUri"` attribute in the `web.config`, ASP.NET does not send the ASP.NET Session ID back as a cookie. Instead, it modifies every URL to include the Session ID just before the requested page:

```
<sessionState mode="SQLServer" cookieless="UseUri" sqlConnectionString="data
source=127.0.0.1;user id=sa;password=Wrox"></sessionState>
```

Notice that the Session ID appears in the URL as if it were a directory of its own situated between the actual Web site virtual directory and the page. With this change, server-side user controls such as the `HyperLink` control, used in Listing 22-1, have their properties automatically modified. The link in Listing 22-1 could have been hard-coded as HTML directly in the Designer, but then ASP.NET could not modify the target URL shown in Figure 22-8.

The Session ID is a string that contains only the ASCII characters allowed in a URL. That makes sense when you realize that moving from a cookie-based Session-State system to a cookieless system requires putting that Session State value in the URL.

Notice in Figure 22-8 that the request URL contains a Session ID within parentheses. One disadvantage to cookieless Sessions is how easily they can be tampered with. Certainly, cookies can be tampered with using HTTP sniffers, but URLS can be edited by anyone. The only way Session State is maintained is if *every* URL includes the Session ID in this way.


Step 2 - Windows Internet Explorer
_app/(S(vmgylh55rhnmezqmx40uv5jl))/Step2.aspx
Google
Step 2
Page
Tools
Done
Internet | Protected Mode: Off
100%

Scott Hanselman

Request Details

Session Id: vmgylh55rhnmezqmx40uv5jl **Request Type:** POST

Figure 22-8

Additionally, all URLS *must* be relative. Remember that the Session ID appears as if it were a directory. The Session is lost if an absolute URL such as `/myapp/retrieve.aspx` is invoked. If you are generating URLs on the server side, use `HttpResponse.ApplyAppPathModifier()`. It changes a URL when the Session ID is embedded, as shown here:

```
Response.Write(Response.ApplyAppPathModifier("foo/bar.aspx"));
```

The previous line generates a URL similar to the following:

```
/myapp/ (S(avkbnbml4n1n5mi5dmfqnu45))/foo/bar.aspx
```

Notice that not only was session information added to the URL, but it was also converted from a relative URL to an absolute URL, including the application's virtual directory. This method can be useful when you need to use `Response.Redirect` or build a URL manually to redirect from an HTTP page to an HTTPS page while still maintaining cookieless session state.

Choosing the Correct Way to Maintain State

Now that you're familiar with the variety of options available for maintaining state in ASP.NET 2.0, here's some real-world advice from production systems. The In-Process (InProc) Session provider is the fastest method, of course, because everything held in memory is a live object reference. This provider is held in the `HttpApplication's` cache and, as such, it is susceptible to application recycles. If you use Windows 2000 Server or Windows XP, the `aspnet_wp.exe` process manages the ASP.NET HTTP pipeline. If you're running Windows 2003 Server or Vista, `w3wp.exe` is the default process that hosts the runtime.

You must find a balance between the robustness of the out-of-process state service and the speed of the in-process provider. In our experience, the out-of-process state service is usually about 15 percent slower than the in-process provider because of the serialization overhead and marshaling. SQL Session State is about 25 percent slower than InProc. Of course your mileage will likely vary. Don't let these numbers concern you too much. Be sure to do scalability testing on your applications before you panic and make inappropriate decisions.

It's worth saying again: We recommend that all developers use Out-Of-Process Session State during development, even if this is not the way your application will be deployed. Forcing yourself to use the Out-Of-Process provider enables you to catch any potential problems with custom objects that do not carry the `Serializable` attribute. If you design your entire site using the In-Process provider and then discover, late in the project, that requirements force you to switch to the SQL or Out-Of-Process providers, you have no guarantee that your site will work as you wrote it. Developing with the Out-Of-Process provider gives you the best of both worlds and does not affect your final deployment method. Think of it as an insurance policy that costs you nothing upfront.

The Application Object

The `Application` object is the equivalent of a bag of global variables for your ASP.NET application. Global variables have been considered harmful for many years in other programming environments, and ASP.NET is no different. You should give some thought to what you want to put in the `Application` object and why. Often, the more flexible `Cache` object that helps you control an object's lifetime is the more useful. Caching is discussed in depth in Chapter 23.

The `Application` object is not global to the machine; it's global to the `HttpApplication`. If you are running in the context of a Web farm, each ASP.NET application on each Web server has its own `Application` object. Because ASP.NET applications are multithreaded and are receiving requests that are being handled by your code on multiple threads, access to the `Application` object should be managed using the `Application.Lock` and `Application.Unlock` methods. If your code doesn't call Unlock directly (which it should, shame on you) the lock is removed implicitly at the end of the `HttpRequest` that called Lock originally.

This small example shows you how to lock the `Application` object just before inserting an object. Other threads that might be attempting to write to the `Application` will wait until it is unlocked. This example assumes there is an integer already stored in `Application` under the key `GlobalCount`.

VB

```
Application.Lock()
Application("GlobalCount") = CType(Application("GlobalCount"), Integer) + 1
Application.UnLock()
```

C#

```
Application.Lock();
Application["GlobalCount"] = (int)Application["GlobalCount"] + 1;
Application.UnLock();
```

Object references can be stored in the `Application`, as in the `Session`, but they must be cast back to their known types when retrieved (as shown in the preceding sample code).

QueryStrings

The URL, or QueryString, is the ideal place for navigation-specific — not user-specific — data. The QueryString is the most hackable element on a Web site, and that fact can work for you or against you. For example, if your navigation scheme uses your own page IDs at the end of a query string (such as `/localhost/mypage.aspx?id = 54`) be prepared for a user to play with that URL in his browser, and try every value for `id` under the sun. Don't blindly cast `id` to an int, and if you do, have a plan if it fails. A good idea is to return `Response.StatusCode=404` when someone changes a URL to an unreasonable value. Another fine idea that Amazon.com implemented was the *Smart 404*. Perhaps you've seen these: They say ''Sorry you didn't find what you're looking for. Did you mean _____?''

Remember, your URLs are the first thing your users may see, even before they see your HTML. *Hackable* URLs — hackable even by my mom — make your site more accessible. Which of these URLs is friendlier and more hackable (for the *right* reason)?

```
http://reviews.cnet.com/Philips_42PF9996/4505-6482_7-31081946.html?tag=cnetfd.sd
```

or

```
http://www.hanselman.com/blog/CategoryView.aspx?category=Movies
```

Cookies

Do you remember the great cookie scare of 1997? Most users weren't quite sure just what a cookie was, but they were all convinced that cookies were evil and were storing their personal information. Back then, it was likely personal information was stored in the cookie! Never, ever store sensitive information, such as a user ID or password, in a cookie. Cookies should be used to store only non-sensitive information, or information that can be retrieved from an authoritative source. Cookies shouldn't be trusted, and their contents should be able to be validated. For example, if a Forms Authentication cookie has been tampered with, the user is logged out and an exception is thrown. If an invalid Session ID cookie is passed in for an expired Session, a new cookie can be assigned.

When you store information in cookies, remember that it's quite different from storing data in the `Session` object:

- ❑ Cookies are passed back and forth on *every* request. That means you are paying for the size of your cookie during *every* HTTP GET and HTTP POST.
- ❑ If you have ten 1-pixel spacer GIFs on your page used for table layouts, the user's browser is sending the same cookie *eleven* times: once for the page itself, and once for each spacer GIF, even if the GIF is already cached.
- ❑ Cookies can be stolen, sniffed, and faked. If your code counts on a cookie's value, have a plan in your code for the inevitability that cookie will get corrupted or be tampered with.
- ❑ What is the expected behavior of your application if a cookie doesn't show? What if it's 4096 bytes? Be prepared. You should design your application around the ''principle of least surprise.'' Your application should attempt to heal itself if cookies are found missing or if they are larger than expected.
- ❑ Think twice before Base64 encoding anything large and placing it in a cookie. If your design depends on this kind of technique, rethink using either the Session or another backing-store.

PostBacks and Cross-Page PostBacks

In classic ASP, in order to detect logical events such as a button being clicked, developers had to inspect the `Form` collection of the `Request` object. Yes, a button was clicked in the user's browser, but no object model was built on top of stateless HTTP and HTML. ASP.NET 1.*x* introduced the concept of the postback, wherein a server-side event was raised to alert the developer of a client-side action. If a button is clicked on the browser, the Form collection is POSTed back to the server, but now ASP.NET allowed the developer to write code in events such as `Button1_Click` and `TextBox1_Changed`.

However, this technique of posting *back* to the same page is counter-intuitive, especially when you are designing user interfaces that aim to create wizards to give the user the sense of forward motion.

This chapter is about all aspects of state management. Postbacks and cross-page postbacks, however, are covered extensively in Chapter 3 so this chapter touches on them only in the context of state management. Postbacks were introduced in ASP.NET 1.*x* to provide an eventing subsystem for Web development. It was inconvenient to have only single-page postbacks in 1.*x*, however, and that caused many developers to store small objects in the `Session` on a postback and then redirect to the next page to pick up the stored data. With cross-page postbacks, data can be posted ''forward'' to a different page, often obviating the need for storing small bits of data that could be otherwise passed directly.

ASP.NET 2.0 and above includes the notion of a `PostBackUrl` to all the Button controls including LinkButton and ImageButton. The `PostBackUrl` property is both part of the markup when a control is presented as part of the ASPX page, as seen in the following, and is a property on the server-side component that's available in the code-behind:

```
<asp:Button PostBackUrl="url" ..>
```

When a button control with the `PostBackUrl` property set is clicked, the page does not post back to itself; instead, the page is posted to the URL assigned to the button control's `PostBackUrl` property. When a cross-page request occurs, the `PreviousPage` property of the current `Page` class holds a reference to the page that caused the postback. To get a control reference from the `PreviousPage`, use the `Controls` property or use the `FindControl` method.

Create a fresh site with a `Default.aspx` (as shown in Listing 22-8). Put a `TextBox` and a `Button` on it, and set the `Button` `PostBackUrl` property to `Step2.aspx`. Then create a `Step2.aspx` page with a single `Label` and add a `Page_Load` handler by double-clicking the HTML Designer.

Listing 22-8: Cross-page postbacks

Default.aspx

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/
xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Cross-page PostBacks</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
        <asp:Button ID="Button1" Runat="server" Text="Button"
```

```
                PostBackUrl="~/Step2.aspx" />
        </div>
        </form>
</body>
</html>
```

Step2.aspx

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Step 2</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
    </div>
    </form>
</body>
 </html>
```

VB — Step2.aspx.vb

```
Partial Class Step2
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
             Handles Me.Load

        If PreviousPage IsNot Nothing AndAlso PreviousPage.IsCrossPagePostBack Then
            Dim text As TextBox = _
                    CType(PreviousPage.FindControl("TextBox1"), TextBox)
            If text IsNot Nothing Then
                Label1.Text = text.Text
            End If
        End If

    End Sub

End Class
```

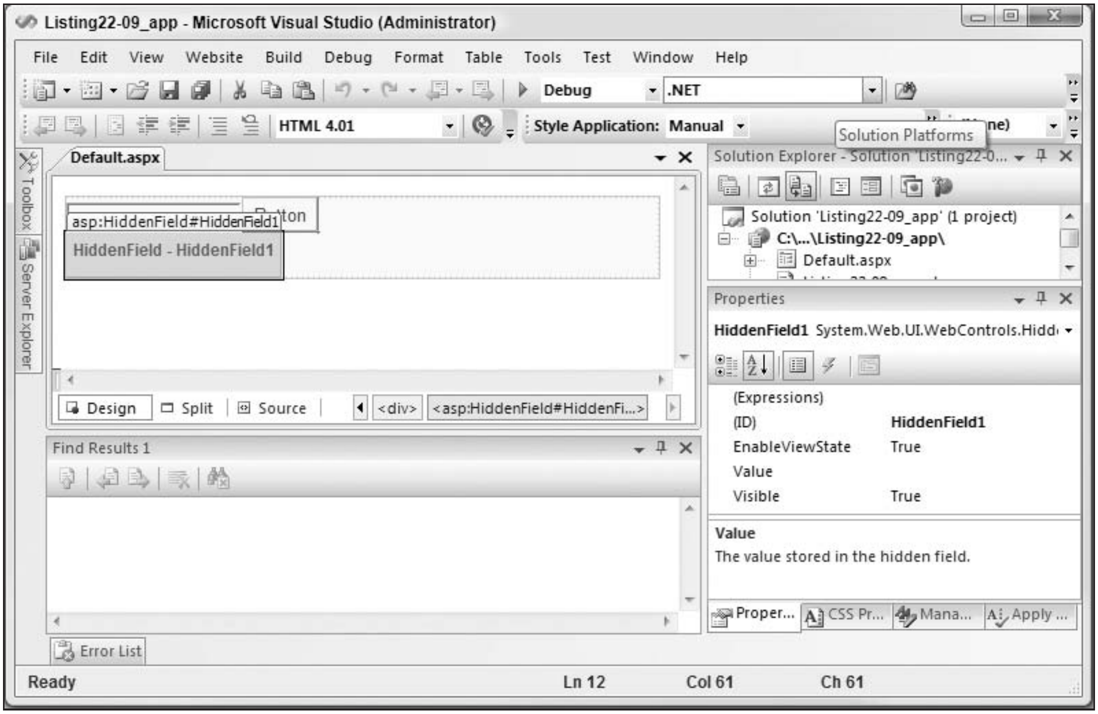
CS — Step2.aspx.cs

```
using System;
using System.Web.UI.WebControls;

public partial class Step2 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (PreviousPage != null && PreviousPage.IsCrossPagePostBack)
        {
            TextBox text = PreviousPage.FindControl("TextBox1") as TextBox;
            if (text != null)
            {
                Label1.Text = text.Text;
            }
        }
    }

}
```

In Listing 22-8, `Default.aspx` posts *forward* to `Step2.aspx`, which can then access the `Page.PreviousPage` property and retrieve a populated instance of the `Page` that caused the postback. A call to `FindControl` and a cast retrieves the `TextBox` from the previous page and copies its value into the Label of `Step2.aspx`.

Hidden Fields, ViewState, and ControlState

Hidden input fields such as `<input type=""hidden" name="foo">` are sent back as name/value pairs in a Form POST exactly like any other control, except they are not rendered. Think of them as hidden text boxes. Figure 22-9 shows a HiddenField control on the Visual Studio Designer with its available properties. Hidden fields are available in all versions of ASP.NET.


Listing22-09_app - Microsoft Visual Studio (Administrator)
File
Edit
View
Website
Build
Debug
Format
Table
Tools
Test
Window
Help
Debug
.NET
HTML 4.01
Style Application: Manual
Solution Platforms
Default.aspx
Toolbox
Server Explorer
asp:HiddenField#HiddenField1
Button
HiddenField - HiddenField1
Design
Split
Source
<div>
<asp:HiddenField#HiddenFi...>
Find Results 1
Error List
Solution Explorer - Solution 'Listing22-0...
Solution 'Listing22-09_app' (1 project)
C:\...\Listing22-09_app\
Default.aspx
Properties
HiddenField1 System.Web.UI.WebControls.Hidd
Proper...
CSS Pr...
Mana...
Apply ...
Ready
Ln 12
Col 61
Ch 61

(Expressions)
(ID) HiddenField1
EnableViewState True
Value
Visible True

Value
The value stored in the hidden field.

Figure 22-9

ViewState, on the other hand, exposes itself as a collection of key/value pairs like the `Session` object, but renders itself as a hidden field with the name `"__VIEWSTATE"` like this:

```
<input type="hidden" name="__VIEWSTATE" value="/AAASSDAS...Y/lOI=" />
```

Any objects put into the ViewState must be marked `Serializable`. ViewState serializes the objects with a special binary formatter called the LosFormatter. LOS stands for limited object serialization. It serializes any kind of object, but it is optimized to contain strings, arrays, and hashtables.

To see this at work, create a new page and drag a `TextBox`, `Button`, and `HiddenField` onto it. Double-click in the Designer to create a `Page_Load` and include the code from Listing 22-9. This example adds a string to `HiddenField.Value`, but adds an instance of a `Person` to the `ViewState` collection. This listing illustrates that while ViewState is persisted in a single HTML `TextBox` on the client, it can contain both simple types such as strings, and complex types such as `Person`. This technique has been around since ASP.NET 1.*x* and continues to be a powerful and simple way to persist small pieces of data without utilizing server resources.

Listing 22-9: Hidden fields and ViewState

ASPX

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Hidden Fields and ViewState</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
        <asp:Button ID="Button1" Runat="server" Text="Button"  />
        <asp:HiddenField ID="HiddenField1" Runat="server" />
    </div>
    </form>
</body>
</html>
```

VB

```
<Serializable> _
Public Class Person
    Public firstName As String
    Public lastName As String
End Class

Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
            Handles Me.Load

        If Not Page.IsPostBack Then
            HiddenField1.Value = "foo"
            ViewState("AnotherHiddenValue") = "bar"

            Dim p As New Person
            p.firstName = "Scott"
            p.lastName = "Hanselman"
            ViewState("HiddenPerson") = p
        End If

    End Sub

End Class
```

C#

```
using System;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

[Serializable]
public class Person
{
    public string firstName;
    public string lastName;
}

public partial class _Default : System.Web.UI.Page
{

    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            HiddenField1.Value = "foo";
            ViewState["AnotherHiddenValue"] = "bar";

            Person p = new Person();
            p.firstName = "Scott";
            p.lastName = "Hanselman";
            ViewState["HiddenPerson"] = p;
        }
    }
}
```

In Listing 22-9, a string is added to a `HiddenField` and to the `ViewState` collection. Then a `Person` instance is added to the `ViewState` collection with another key. A fragment of the rendered HTML is shown in the following code:

```
<form method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE"
value="/wEPDwULLTIxMjQ3OTEzODcPFgQeEkFub3RoZXJIaWRkZW5WYWx1ZQUDYmFyHgxIaWRkZW5QZXJz
b24ypwEAAQAAAP////8BAAAAAAAAAAwCAAAAP3ZkcTVqYzdxLCBWZXJzaW9uPTAuMC4wLjAsIEN1bHR1cmU
9bmV1dHJhbCwgUHVibGljS2V5VG9rZW49bnVsbAUBAAAAE0RlZmF1bHRfYXNweCtQZXJzb24CAAAACWZpcn
N0TmFtZQhsYXN0TmFtZQEBAgAAAAYDAAAABVNjb3R0BgQAAAAJSGFuc2VsbWFuC2RkI/CLauUviFo58BF8v
pSNsjY/lOI=" />
</div>
```

```
        <div>
            <input name="TextBox1" type="text" id="TextBox1" />
            <input type="submit" name="Button1" value="Button" id="Button1" />
            <input type="hidden" name="HiddenField1" id="HiddenField1" value="foo" />
        </div>
    </form>
```

Notice that the `ViewState` value uses only valid ASCII characters to represent all its contents. Don't let the sheer mass of it fool you. It is big and it appears to be opaque. However, it's just a hidden text box and is automatically POSTed back to the server. The entire `ViewState` collection is available to you in the `Page_Load`. The value of the `HiddenField` is stored as plain text.

Neither ViewState nor Hidden Fields are acceptable for any kind of sensitive data.

People often complain about the size of ViewState and turn if off completely without realizing its benefits. ASP.NET 2.0 cut the size of serialized ViewState nearly in half. You can find a number of tips on using ViewState on my blog by Googling for ''Hanselman ViewState''. Fritz Onion's free ViewStateDecoder tool from `www.pluralsight.com` is a great way to gain insight into what's stored in your pages' ViewState. Note also Nikhil Kotari's detailed blog post on ViewState improvements at `www.nikhilk.net/ViewStateImprovements.aspx`.

By default, the ViewState field is sent to the client with a *salted hash* to prevent tampering. Salting means that the ViewState's data has a unique value appended to it before it's encoded. As Keith Brown says ''Salt is just one ingredient to a good stew.'' The technique used is called HMAC, or hashed message authentication code. As shown in the following code, you can use the `<machineKey>` element of the `web.config file` to specify the `validationKey`, as well as the algorithm used to protect ViewState. This section of the file and the `decryptionKey` attribute also affect how Forms Authentication cookies are encrypted (see Chapter 21 for more on forms authentication).

```
<machineKey validationKey="AutoGenerate,IsolateApps"
 decryptionKey="AutoGenerate,IsolateApps" validation="SHA1" />
```

If you are running your application in a Web farm, `<validationKey>` and `<decryptionKey>` have to be manually set to the same value. Otherwise, ViewState generated from one machine could be POSTed back to a machine in the farm with a different key! The keys should be 128 characters long (the maximum) and generated totally by random means. If you add `IsolateApps` to these values, ASP.NET generates a unique encrypted key for each application using each application's application ID.

I like to use security guru Keith Brown's GenerateMachineKey tool, which you can find at `www.pluralsight.com/tools.aspx`, *to generate these keys randomly.*

The `validation` attribute can be set to SHA1 or MD5 to provide tamper-proofing, but you can include added protection by encrypting ViewState as well. In ASP.NET 1.1 you can encrypt ViewState only by using the value 3DES in the `validation` attribute, and ASP.NET 1.1 will use the key in the `decryptionKey` attribute for encryption. However, ASP.NET 2.0 adds a new decryption attribute that is used exclusively for specifying the encryption and decryption mechanisms for forms authentication tickets, and the `validation` attribute is used exclusively for ViewState, which can now be encrypted using 3DES or AES and the key stored in the `validationKey` attribute.

ASP.NET 2.0 also adds the `ViewStateEncryptionMode` attribute to the `<pages>` configuration element with two possible values, `Auto` or `Always`. Setting the attribute to `Always` will force encryption of ViewState, whereas setting it to `Auto` will encrypt ViewState only if a control requested encryption using the new `Page.RegisterRequiresViewStateEncryption` method.

Added protection can be applied to ViewState by setting `Page.ViewStateUserKey` in the `Page_Init` to a unique value such as the user's ID. This must be set in `Page_Init` because the key should be provided to ASP.NET before ViewState is loaded or generated. For example:

```
protected void Page_Init (Object sender, EventArgs e)
{
    if (User.Identity.IsAuthenticated)
        ViewStateUserKey = User.Identity.Name;
}
```

When optimizing their pages, ASP.NET programmers often disable ViewState for many controls when that extra bit of state isn't absolutely necessary. However, in ASP.NET 1.*x*, disabling ViewState was a good way to break many third-party controls, as well as the included DataGrid's sorting functionality. ASP.NET now includes a second, parallel ViewState-like collection called `ControlState`. This dictionary can be used for round-tripping crucial information of limited size that should not be disabled even when ViewState is. You should only store data in the `ControlState` collection that is absolutely critical to the functioning of the control.

Recognize that ViewState, and also ControlState, although not secure, is a good place to store small bits of a data and state that don't quite belong in a cookie or the `Session` object. If the data that must be stored is relatively small and local to that specific instance of your page, ViewState is a much better solution than littering the `Session` object with lots of transient data.

Using HttpContext.Current.Items for Very Short-Term Storage

The `Items` collection of `HttpContext` is one of ASP.NET's best-kept secrets. It is an `IDictionary` key/value collection of objects that's shared across the life of a single `HttpRequest`. That's a *single* `HttpRequest`. Why would you want to store state for such a short period of time? Consider these reasons:

- **When you share content between IHttpModules and IHttpHandlers:** If you write a custom IHttpModule, you can store context about the user for use later in a page.
- **When you communicate between two instances of the same UserControl on the same page:** Imagine you are writing a UserControl that serves banner ads. Two instances of the same control could select their ads from `HttpContext.Items` to prevent showing duplicates on the same page.
- **When you store the results of expensive calls that might otherwise happen twice or more on a page:** If you have multiple UserControls that each show a piece of data from a large, more

expensive database retrieval, those UserControls can retrieve the necessary data from `HttpContext.Items`. The database is hit only once.

- **When individual units within a single HttpRequest need to act on the same or similar data:** If the lifetime of your data is just one request, consider using `HttpContext.Items`, as a short term cache.

The `Items` collection holds objects, just like many of the collections that have been used in this chapter. You need to cast those objects back to their specific type when they are retrieved.

Within a Web-aware Database Access Layer, per-request caching can be quickly implemented with the simple coding pattern shown here. Note that this sample code is a design pattern and there is no `MyData` class; it's for illustration.

VB

```
Public Shared Function GetExpensiveData(ID As Integer) As MyData
  Dim key as string = "data" & ID.ToString()
  Dim d as MyData = _
      CType(HttpContext.Current.Items(key), MyData)
  If d Is Nothing Then
    d = New Data()
    'Go to the Database, do whatever...
    HttpContext.Current.Items(key) = d
  End If
  Return d
End Function
```

C#

```
public static MyData GetExpensiveData(int ID)
{
  string key = "data" + ID.ToString();
  MyData d = (MyData) HttpContext.Current.Items[key];
  if (d == null)
  {
    d = new Data();
    //Go to the Database, do whatever...
    HttpContext.Current.Items[key] = d;
  }
  return d;
}
```

This code checks the `Items` collection of the current `HttpContext` to see if the data is already there. If not, the data is retrieved from the appropriate backing store and then stored in the `Items` collection. Subsequent calls to this function within the same `HttpRequest` receive the already-cached object.

As with all optimizations and caching, premature optimization is the root of all evil. Measure your need for caching, and measure your improvements. Don't cache just because it *feels right*; cache because it makes sense.

Summary

This chapter explored the many ways to manage State within your ASP.NET application. The `Session` object and its providers offer many choices. Each has its own pros and cons for managing state in the form of object references and serialized objects in a way that can be made largely transparent to the application. Server-side Session state data can have its unique identifying key stored in a cookie or the key can be carried along in the URL. Cookies can also be used independently to store small amounts of data and persist it between visits, albeit in much smaller amounts and with simpler types. Hidden fields, ViewState, ControlState, postbacks, and new cross-page postbacks offer new possibilities for managing small bits of state within a multi-page user experience. `HttpContext.Current.Items` offers a perfect place to hold transient state, living the life of only a single `HttpRequest`. QueryStrings are an old standby for holding non-private state that is appropriate for navigation.

ASP.NET has improved on ASP.NET 1.*x*'s state management options with a flexible Session State Provider module, the addition of Control State for user controls, and cross-page postbacks for a more mature programming model.

23

Caching

Performance is a key requirement for any application or piece of code that you develop. The browser helps with client-side caching of text and images, whereas the server-side caching you choose to implement is vital for creating the best possible performance. *Caching* is the process of storing frequently used data on the server to fulfill subsequent requests. You will discover that grabbing objects from memory is much faster than re-creating the Web pages or items contained in them from scratch each time they are requested. Caching increases your application's performance, scalability, and availability. The more you fine-tune your application's caching approach, the better it performs.

This chapter focuses on caching, including the SQL invalidation caching capabilities that ASP.NET provides. This chapter takes a close look at this unique aspect of caching. When you are using SQL cache invalidation, if the result set from SQL Server changes, the output cache can be triggered to change automatically. This ensures that the end user always sees the latest result set, and the data presented is never stale. After introducing SQL cache invalidation, this chapter also covers other performance enhancements. It discusses the new Post-Cache Substitution feature, which caches entire pages while dynamically replacing specified bits of content. Lastly, this chapter covers a new capability that enables a developer to create custom dependencies.

Caching

There are several ways to deal with caching in ASP.NET. First, you can cache an entire HTTP response (the entire Web page) using a mechanism called output caching. Two other methods are partial page caching and data caching. The following sections describe these methods.

Output Caching

Output caching is a way to keep the dynamically generated page content in the server's memory or disk for later retrieval. This type of cache saves post-rendered content so it won't have to be regenerated again the next time it's requested. After a page is cached, it can be served up again when any subsequent requests are made to the server. You apply output caching by inserting an `OutputCache` page directive at the top of an `.aspx` page, as follows:

```
<%@ OutputCache Duration="60" VaryByParam="None" %>
```

The `Duration` attribute defines the number of seconds a page is stored in the cache. The `VaryByParam` attribute determines which versions of the page output are actually cached. You can generate different responses based on whether an HTTP-POST or HTTP-GET response is required. Other than the attributes for the `OutputCache` directive, ASP.NET includes the `VaryByHeader`, `VaryByCustom`, `VaryByControl`, and `Location` attributes. Additionally, the `Shared` attribute can affect UserControls, as you'll see later.

Caching in ASP.NET is implemented as an `HttpModule` that listens to all `HttpRequest`s that come through the ASP.NET worker process. The `OutputCacheModule` listens to the application's `ResolveRequestCache` and `UpdateRequestCache` events, handles cache hits and misses, and returns the cached HTML, bypassing the Page Handler if need be.

VaryByParam

The `VaryByParam` attribute can specify which `QueryString` parameters cause a new version of the page to be cached:

```
<%@ OutputCache Duration="90" VaryByParam="pageId;subPageId" %>
```

For example, if you have a page called `navigation.aspx` that includes navigation information in the `QueryString`, such as `pageId` and `subPageId`, the `OutputCache` directive shown here caches the page for every different value of `pageId` and `subPageId`. In this example, the number of pages is best expressed with an equation:

```
cacheItems = (num of pageIds) * (num of subPageIds)
```

where `cacheItems` is the number of rendered HTML pages that would be stored in the cache. Pages are cached only after they're requested and pass through the `OutputCacheModule`. The maximum amount of cache memory in this case is used only after every possible combination is visited at least once. Although these are just *potential* maximums, creating an equation that represents your system's potential maximum is an important exercise.

If you want to cache a new version of the page based on any differences in the `QueryString` parameters, use `VaryByParam = "*"`, as in the following code.

```
<%@ OutputCache Duration="90" VaryByParam="*" %>
```

It's important to ''do the math'' when using the VaryBy attributes. For example, you could add `VaryByHeader` and cache a different version of the page based on the browser's reported `User-Agent` HTTP Header.

```
<%@ OutputCache Duration="90" VaryByParam="*" VaryByHeader="User-Agent"%>
```

The `User-Agent` identifies the user's browser type. ASP.NET can automatically generate different renderings of a given page that are customized to specific browsers, so it makes sense in many cases to save these various renderings in the cache. A Firefox user might have slightly different HTML than an IE user so we don't want to send all users the exact same post-rendered HTML. Literally dozens, if not hundreds, of `User-Agent` strings exist in the wild because they identify more than just the browser type;

this `OutputCache` directive could multiply into thousands of different versions of this page being cached, depending on server load. In this case, you should measure the cost of the caching against the cost of re-creating the page dynamically.

Always cache what will give you the biggest performance gain, and prove that assumption with testing. Don't "cache by coincidence" using attributes like `VaryByParam = "*"`. A common rule of thumb is to cache the least possible amount of data at first and add more caching later if you determine a need for it. Remember that the server memory is a limited resource so you may want configure the use of disk caching in some cases. Be sure to balance your limited resources with security as a primary concern; don't put sensitive data on the disk.

VaryByControl

`VaryByControl` can be a very easy way to get some serious performance gains from complicated UserControls that render a lot of HTML that doesn't change often. For example, imagine a UserControl that renders a ComboBox showing the names of all the countries in the world. Perhaps those names are retrieved from a database and rendered in the combo box as follows:

```
<%@ OutputCache Duration="2592000" VaryByControl="comboBoxOfCountries" %>
```

Certainly the names of the world's countries don't change that often, so the `Duration` might be set to a month (in seconds). The rendered output of the UserControl is cached, allowing a page using that control to reap performance benefits of caching the control while the page itself remains dynamic.

VaryByCustom

Although the `VaryBy` attributes offer a great deal of power, sometimes you need more flexibility. If you want to take the `OutputCache` directive from the previous navigation example and cache by a value stored in a cookie, you can add `VaryByCustom`. The value of `VaryByCustom` is passed into the `GetVaryByCustomString` method that can be added to the `Global.asax.cs`. This method is called every time the page is requested, and it is the function's responsibility to return a value.

A different version of the page is cached for each unique value returned. For example, say your users have a cookie called `Language` that has three potential values: en, es, and fr. You want to allow users to specify their preferred language, regardless of their language reported by their browser. `Language` also has a fourth potential value — it may not exist! Therefore, the `OutputCache` directive in the following example caches many versions of the page, as described in this equation:

```
cacheItems = (num of pageIds) * (num of subPageIds) * (4 possible Language values)
```

To summarize, suppose there were 10 potential values for `pageId`, five potential `subPageId` values for each `pageId`, and 4 possible values for `Language`. That adds up to 200 different potential cached versions of this single navigation page. This math isn't meant to scare you away from caching, but you should realize that with great (caching) power comes great responsibility.

The following `OutputCache` directive includes `pageId` and `subPageId` as values for `VaryByParam`, and `VaryByCustom` passes in the value of `"prefs"` to the `GetVaryByCustomString` callback function in Listing 23-1:

```
<%@ OutputCache Duration="90" VaryByParam="pageId;subPageId" VaryByCustom="prefs"%>
```

Caching in ASP.NET involves a tradeoff between CPU and memory: how hard is it to make this page, versus whether you can afford to hold 200 versions of it. If it's only 5 KB of HTML, a potential megabyte of memory could pay off handsomely versus thousands and thousands of database accesses. Since most pages will hit the database at least once during a page cycle, every page request served from the cache saves you a trip to the database. Efficient use of caching can translate into cost savings if fewer database servers and licenses are needed.

The code in Listing 23-1 returns the value stored in the `Language` cookie. The `arg` parameter to the `GetVaryByCustomString` method contains the string `"prefs"`, as specified in `VaryByCustom`.

Listing 23-1: GetVaryByCustomString callback method in the HttpApplication

VB

```
Overrides Function GetVaryByCustomString(ByVal context As HttpContext, _
        ByVal arg As String) As String
    If arg.ToLower() = "prefs" Then
        Dim cookie As HttpCookie = context.Request.Cookies("Language")
        If cookie IsNot Nothing Then
            Return cookie.Value
        End If
    End If
    Return MyBase.GetVaryByCustomString(context, arg)
End Function
```

C#

```
public override string GetVaryByCustomString(HttpContext context, string arg)
{
    if(arg.ToLower() == "prefs")
    {
      HttpCookie cookie = context.Request.Cookies["Language"];
      if(cookie != null)
      {
          return cookie.Value;
      }
    }
    return base.GetVaryByCustomString(context, arg);
}
```

The `GetVaryByCustomString` method in Listing 23-1 is used by the `HttpApplication` in `Global.asax.cs` and will be called for every page that uses the `VaryByCustom` `OutputCache` directive. If your application has many pages that use `VaryByCustom`, you can create a switch statement and a series of helper functions to retrieve whatever information you want from the user's HttpContext and to generate unique values for cache keys.

Partial Page (UserControl) Caching

Similar to output caching, *partial page caching* enables you to cache only specific blocks of a Web page. You can, for example, cache only the center of the page the user sees. Partial page caching is achieved with the caching of user controls so you can build your ASP.NET pages to utilize numerous user controls and then apply output caching to the selected user controls. This, in essence, caches only the parts of the page that you want, leaving other parts of the page outside the reach of caching. This is a nice feature and, if done correctly, it can lead to pages that perform better. This requires a modular design to be planned up front so you can partition the components of the page into logical units composed of user controls.

Typically, UserControls are designed to be placed on multiple pages to maximize reuse of common functionality. However, when these UserControls (ASCX files) are cached with the `@OutputCache` directive's default attributes, they are cached on a per-page basis. That means that even if a UserControl outputs the identical HTML when placed on `pageA.aspx` as it does when placed on `pageB.aspx`, its output is cached twice. By enabling the `Shared = "true"` attribute, the UserControl's output can be shared among multiple pages and on sites that make heavy use of shared UserControls:

```
<%@ OutputCache Duration="300" VaryByParam="*" Shared="true" %>
```

The resulting memory savings can be surprisingly large since you only cache one copy of the post-rendered user control instead of caching a copy for each page. As with all optimizations, you need to test both for correctness of output as well as memory usage.

> **If you have an ASCX UserControl using the `OutputCache` directive, remember that the UserControl *exists only for the first request*. If a UserControl has its HTML retrieved from the `OutputCache`, the control doesn't really exist on the ASPX page. Instead, a PartialCachingControl is created that acts as a proxy or *ghost* of that control.**

Any code in the ASPX page that requires a UserControl to be constantly available will fail if that control is reconstituted from the `OutputCache`. So be sure to always check for this type of caching before using any control. The following code fragment illustrates the kind of logic required when accessing a potentially cached UserControl:

VB

```
Protected Sub Page_Load()
    If Not PossiblyCachedUserControl is Nothing Then
       " Place code manipulating PossiblyCachedUserControl here.
    End If
End Sub
```

C#

```
protected void Page_Load()
{
    if (PossiblyCachedUserControl != null)
    {
       // Place code manipulating PossiblyCachedUserControl here.
    }
}
```

Post-Cache Substitution

Output caching has typically been an all-or-nothing proposition. The output of the entire page is cached for later use. However, often you want the benefits of output caching, but you also want to keep a small bit of dynamic content on the page. It would be a shame to cache a page but be unable to output a dynamic ‘‘Welcome, Scott!’’

ASP.NET 2.0 added post-cache substitution as an opportunity to affect the about-to-be-rendered page. A control is added to the page that acts as a placeholder. It calls a method that you specify after the cached content has been returned. The method returns any string output you like, but you should be careful not to abuse the feature. If your post-cache substitution code calls an expensive stored procedure, you could easily lose any performance benefits you might have expected.

Post-cache substitution is an easy feature to use. It gives you two ways to control the substitution:

- ❑ Call the new `Response.WriteSubstitution` method, passing it a reference to the desired substitution method callback.
- ❑ Add a `<asp:Substitution>` control to the page at the desired location, and set its `methodName` attribute to the name of the callback method.

To try this feature, create a new Web site with a `Default.aspx`. Drag a label control and a substitution control to the design surface. The code in Listing 23-2 updates the label to display the current time, but the page is cached immediately and future requests return that cached value. Set the `methodName` property in the substitution control to `GetUpdatedTime`, meaning the name of the static method that is called after the page is retrieved from the cache.

The callback function must be static because the page that is rendered doesn't really exist at this point (an instance of it doesn't). Because you don't have a page instance to work with, this method is limited in its scope. However, the current `HttpContext` is passed into the method, so you have access to the `Session`, `Request`, and `Response`. The string returned from this method is injected into the `Response` in place of the substitution control.

Listing 23-2: Using the substitution control

ASPX

```
<%@ Page Language="C#" CodeFile="Default.aspx.cs" Inherits="_Default" %>
<%@ OutputCache Duration="30" VaryByParam="None" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head >
    <title>Substitution Control</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" Runat="server" Text="Label"></asp:Label>
        <br />
        <asp:Substitution ID="Substitution1" Runat="server"
             methodName="GetUpdatedTime" />
        <br />
    </div>
    </form>
</body>
</html>
```

VB

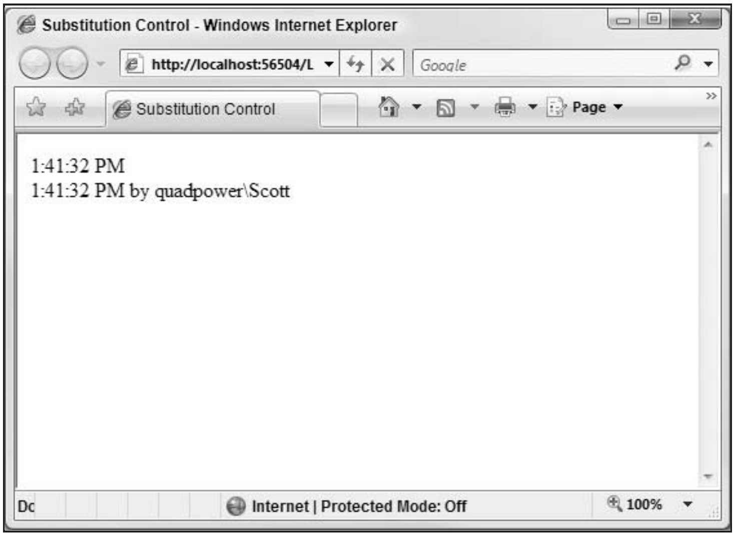
```
Partial Class _Default
        Inherits System.Web.UI.Page
    Public Shared Function GetUpdatedTime(ByVal context As HttpContext) As String
```

```
        Return DateTime.Now.ToLongTimeString() + " by " + _
                context.User.Identity.Name
    End Function

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
            Handles Me.Load
        Label1.Text = DateTime.Now.ToLongTimeString()
    End Sub
End Class
```

C#

```
public partial class _Default : System.Web.UI.Page
{
    public static string GetUpdatedTime(HttpContext context)
    {
        return DateTime.Now.ToLongTimeString() + " by " +
                context.User.Identity.Name;
    }
    protected void Page_Load(object sender, EventArgs e)
    {
        Label1.Text = DateTime.Now.ToLongTimeString();
    }
}
```

The ASPX page in Listing 23-2 has a label and a Post-Cache Substitution Control. The control acts as a placeholder in the spot where you want fresh content injected after the page is returned from the cache. The very first time the page is visited only the label is updated because no cached content is returned. The second time the page is visited, however, the entire page is retrieved from the cache — the page handler isn't called and, consequently, none of the page-level events fire. However, the `GetUpdatedTime` method is called after the cache module completes its work. Figure 23-1 shows the result if the first line is cached and the second line is created dynamically.


Substitution Control - Windows Internet Explorer
http://localhost:56504/L
Google
Substitution Control
Page
Dc
Internet | Protected Mode: Off
100%

```
1:41:32 PM
1:41:32 PM by quadpower\Scott
```

Figure 23-1

HttpCachePolicy and Client-Side Caching

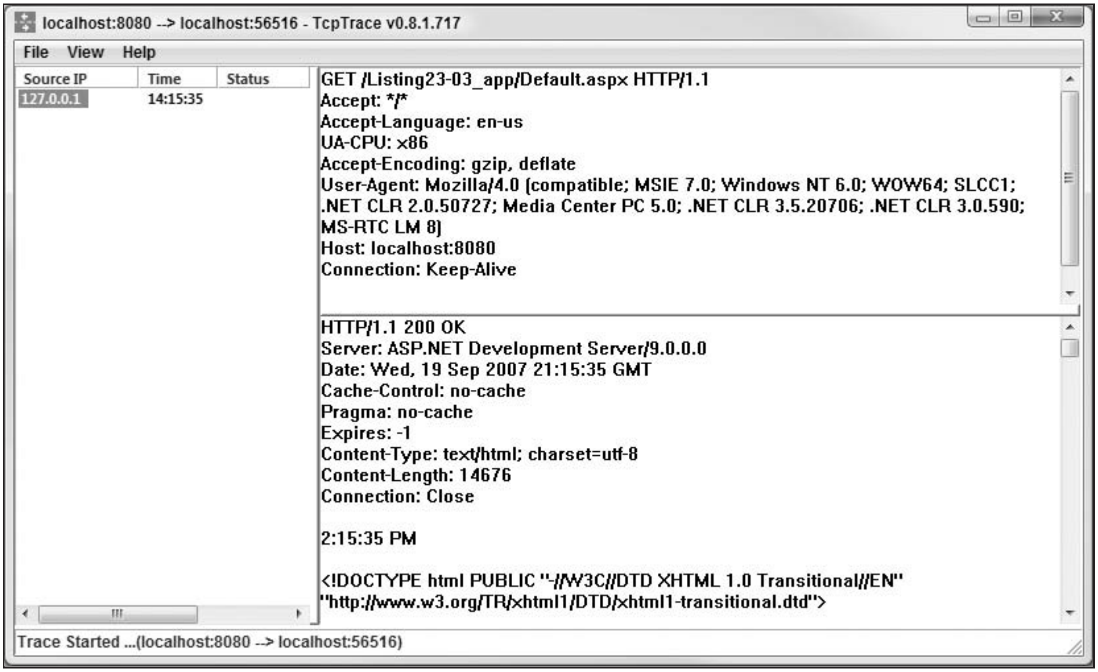
Caching is more than just holding data in memory on the server-side. A good caching strategy should also include the browser and its client-side caches, controlled by the Cache-Control HTTP Header. HTTP Headers are hints and directives to the browser on how to handle a request.

Some people recommend using HTML `<META>` tags to control caching behavior. Be aware that neither the browsers nor routers along the way are obligated to pay attention to these directives. You might have more success using HTTP Headers to control caching.

Because HTTP Headers travel outside the body of the HTTP message, you have several options for viewing them. You can enable tracing (see Chapter 21) and view the Headers from the tracing output. In Figure 23-2 I'm using the free TcpTrace redirector from `PocketSoap.com`.

> **For background information on HTTP headers and controlling caching, see the document RFC 2616: Hypertext Transfer Protocol - HTTP/1.1, available on the World Wide Web Consortium's site at `www.w3c.org`. You might also check out Fiddler at `www.fiddlertool.com/fiddler` and FireBug for Firefox at `www.getfirebug.com`. Commercial tools such as HttpWatch from `www.httpwatch.com` add more features.**

Create a `Default.aspx` that writes the current time in its `Load` event. Now, view the default HTTP Headers used by ASP.NET, as in Figure 23-2 with page-level tracing turned on. Note that one header, `Cache-Control: private`, indicates to routers and other intermediates that this response is intended only for you (private).



Source IP	Time	Status
127.0.0.1	14:15:35	

```
GET /Listing23-03_app/Default.aspx HTTP/1.1
Accept: */*
Accept-Language: en-us
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; WOW64; SLCC1;
.NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.5.20706; .NET CLR 3.0.590;
MS-RTC LM 8)
Host: localhost:8080
Connection: Keep-Alive
```

```
HTTP/1.1 200 OK
Server: ASP.NET Development Server/9.0.0.0
Date: Wed, 19 Sep 2007 21:15:35 GMT
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 14676
Connection: Close

2:15:35 PM

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
Trace Started ...(localhost:8080 --> localhost:56516)
```

Figure 23-2

The `HttpCachePolicy` class gives you an object model for managing client-side state that insulates you from adding HTTP headers yourself. Add the lines from Listing 23-3 to your `Page_Load` to influence the Response's headers and the caching behavior of the browser. This listing tells the browser not to cache this Response in memory nor store it on disk. It also directs the Response to expire immediately.

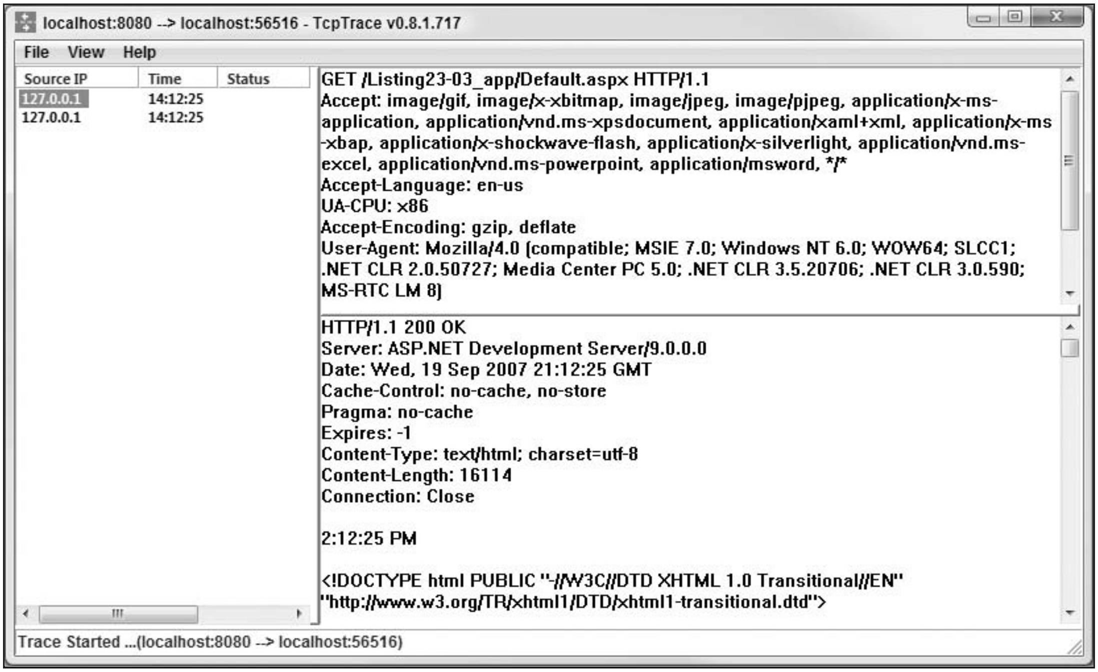
Listing 23-3: Using HTTP Headers to force the browser not to cache on the client-side

VB

```
Protected Sub Page_Load(ByVal sender As Object, _
  ByVal e As System.EventArgs) Handles Me.Load

   Response.Cache.SetCacheability(HttpCacheability.NoCache)
   Response.Cache.SetNoStore()
   Response.Cache.SetExpires(DateTime.MinValue)

   Response.Write(DateTime.Now.ToLongTimeString())
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Cache.SetCacheability(HttpCacheability.NoCache);
    Response.Cache.SetNoStore();
    Response.Cache.SetExpires(DateTime.MinValue);

    Response.Write(DateTime.Now.ToLongTimeString());
}
```



Source IP	Time	Status
127.0.0.1	14:12:25	
127.0.0.1	14:12:25	

```
GET /Listing23-03_app/Default.aspx HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-ms-
application, application/vnd.ms-xpsdocument, application/xaml+xml, application/x-ms
-xbap, application/x-shockwave-flash, application/x-silverlight, application/vnd.ms-
excel, application/vnd.ms-powerpoint, application/msword, */*
Accept-Language: en-us
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; WOW64; SLCC1;
.NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.5.20706; .NET CLR 3.0.590;
MS-RTC LM 8)
```

```
HTTP/1.1 200 OK
Server: ASP.NET Development Server/9.0.0.0
Date: Wed, 19 Sep 2007 21:12:25 GMT
Cache-Control: no-cache, no-store
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 16114
Connection: Close

2:12:25 PM

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Trace Started ...(localhost:8080 --> localhost:56516)

Figure 23-3

Compare the results of running Listing 23-3 in the *before* Figure 23-2 and then in the *after* Figure 23-3. Two new HTTP headers have been injected directing the client's browser and the `Cache-Control` Header has changed to `no-cache, no-store`. The Output Caching HttpModule will respect these HTTP headers, so sending `no-cache`, `no store` to the browser also advises the HttpModule to record the response as a cache miss.

If your ASP.NET application contains a considerable number of relatively static or non–time-sensitive pages, consider what your client-side caching strategy is. It's better to take advantage of the disk space and the memory of your users' powerful client machines rather than burdening your server's limited resources.

Caching Programmatically

Output Caching is a very declarative business. UserControls and Pages can be marked up with `OutputCache` directives and dramatically change the behavior of your site. Declarative caching controls the life cycle of HTML markup, but ASP.NET also includes deep imperative programmatic support for caching objects.

Data Caching Using the Cache Object

Another method of caching is to use the `Cache` object to start caching specific data items for later use on a particular page or group of pages. The `Cache` object enables you to store everything from simple name/value pairs to more complex objects such as datasets and entire `.aspx` pages.

> **Although this is quite similar to Session state, the `Cache` object is shared by all users of the particular web server's app domain that is hosting this application. So if you put a particular item in the `Cache`, all users will be able to see that object. This may not work as expected in a server farm scenario since you can't be assured of which server the user will hit next, and even if there's only one server involved there may be more than one app domain running this application. Also, the server is free to invalidate any cached item at any time if it needs to reclaim some of the memory.**

You use the `Cache` object in the following fashion:

VB

```
Cache("WhatINeedToStore") = myDataSet
```

C#

```
Cache["WhatINeedToStore"] = myDataSet;
```

After an item is in the cache, you can retrieve it later as shown here:

VB

```
Dim ds As New DataSet
ds = CType(Cache("WhatINeedToStore"), DataSet)
```

C#

```
DataSet ds = new DataSet();
ds = (DataSet)Cache["WhatINeedToStore"];
```

Using the `Cache` object is an outstanding way to cache your pages and is, in fact, what the `OutputCache` directive uses under the covers. This small fragment shows the simplest use of the `Cache` object. Simply put an object reference in it. However, the real power of the `Cache` object comes with its capability to invalidate itself. That's where cache dependencies come in.

You must always follow the pattern of testing to see if an item is in the Cache, and if not you need to do whatever processing is necessary to re-create the object. Once re-created you can insert it back into the Cache to become available for the next request.

Controlling the ASP.NET Cache

Ordinarily the default parameters set by ASP.NET for the caching subsystem are appropriate for general use. They are configurable, however, within the machine.config or web.config. These options let you make changes like preventing cached items from expiring when the system is under memory pressure, or turning off item expiration completely. You can set the maximize size the application's private bytes before the cache begins to flush items.

```
<system.web>
  <cache disableMemoryCollection="false"
  disableExpiration="false" privateBytesLimit="0"
  percentagePhysicalMemoryUsedLimit="90"
  privateBytesPollTime="00:02:00" />
...snip...
```

I encourage you to leave the default values as they are unless you've done formal profiling of your application and understand how it utilizes the Cache. More detail on this section can be found on MSDN here: `http://msdn2.microsoft.com/en-us/library/ms228248.aspx`.

Cache Dependencies

Using the `Cache` object, you can store and also invalidate items in the cache based on several different dependencies. In ASP.NET 1.0/1.1, the only possible dependencies were the following:

- ❑ File-based dependencies
- ❑ Key-based dependencies
- ❑ Time-based dependencies

When inserting items into the cache using the `Cache` object, you set the dependencies with the `Insert` method, as shown in the following example:

```
Cache.Insert("DSN", connectionString, _
   New CacheDependency(Server.MapPath("myconfig.xml")))
```

By using a *dependency* when the item being referenced changes, you remove the cache for that item from memory.

Cache Dependencies were improved in ASP.NET 2.0 with the addition of the `AggregateCacheDependency` class, the newly extendable `CacheDependency` class, and the capability to create your own custom `CacheDependency` classes. These three things are discussed in the following sections.

The AggregateCacheDependency Class

The `AggregateCacheDependency` class is like the `CacheDependency` class but it enables you to create an association connecting an item in the cache with many disparate dependencies of *different types*. For example, if you have a cached data item that is built from XML from a file and you also have information from a SQL database table, you can create an `AggregateCacheDependency` with inserted `CacheDependency` objects for each subdependency. To do this, you call `Cache.Insert` and add the `AggregateCacheDependency` instance. For example:

```
Dim agg as new AggregateCacheDependency()
agg.Insert(New CacheDependency(Server.MapPath("myconfig.xml")))
agg.Insert(New SqlCacheDependency("Northwind", "Customers"))
Cache.Insert("DSN", connectionString, agg)
```

Note that `AggregateCacheDependency` is meant to be used with *different* kinds of `CacheDependency` classes. If you simply want to associate one cached item with multiple files, use an overload of `CacheDependency`, as in this example:

VB

```
Cache.Insert("DSN", yourObject, _
    New System.Web.Caching.CacheDependency( _
        New String() _
       { _
         Server.MapPath("foo.xml"), _
         Server.MapPath("bar.xml") _
       } _
    ) _
)
```

C#

```
Cache.Insert("DSN", yourObject,
    new System.Web.Caching.CacheDependency(
        new string[]
        {
            Server.MapPath("foo.xml"),
            Server.MapPath("bar.xml")
        }
    )
);
```

The `AggregateCacheDependency` class is made possible by the new support for extending the previously sealed `CacheDependency` class. You can use this innovation to create your own custom `CacheDependency`.

The Unsealed CacheDependency Class

A big change in caching in ASP.NET 2.0 was that the `CacheDependency` class has been refactored and unsealed (or made overrideable). This allows us tocreate classes that inherit from the `CacheDependency` class and create more elaborate dependencies that are not limited to the `Time`, `Key`, or `File` dependencies.

When you create your own cache dependencies, you have the option to add procedures for such things as Web services data, only-at-midnight dependencies, or textual string changes within a file. The dependencies you create are limited only by your imagination. The unsealing of the `CacheDependency` class puts you in the driver's seat to let you decide when items in the `Cache` need to be invalidated.

Along with the unsealing of the `CacheDependency` class, the ASP.NET team has also built a SQL Server cache dependency — `SqlCacheDependency`. A SQL cache dependency was the caching feature most requested by ASP.NET 1.0/1.1 developers. When a cache becomes invalid because a table changes within the underlying SQL Server, you now know it immediately.

Because `CacheDependency` is now unsealed, you can derive your own custom Cache Dependencies; that's what you do in the next section.

Creating Custom Cache Dependencies

ASP.NET has time-based, file-based, and SQL-based `CacheDependency` support. You might ask yourself why you would write your own `CacheDependency`. Here are a few ideas:

- ❑ Invalidate the cache from the results of an Active Directory lookup query
- ❑ Invalidate the cache upon arrival of an MSMQ or MQSeries message
- ❑ Create an Oracle-specific `CacheDependency`
- ❑ Invalidate the cache using data reported from an XML Web service
- ❑ Update the cache with new data from a Stock Price service

The new version of the `CacheDependency` class, while introducing no breaking changes to existing ASP.NET 1.1 code, exposes three new members and a constructor overload that developers can use:

- ❑ `GetUniqueID`: When overridden, enables you to return a unique identifier for a custom cache dependency to the caller.
- ❑ `DependencyDispose`: Used for disposing of resources used by the custom cache dependency class. When you create a custom cache dependency, you are required to implement this method.
- ❑ `NotifyDependencyChanged`: Called to cause expiration of the cache item dependent on the custom cache dependency instance.
- ❑ New Public Constructor.

Listing 23-4 creates a new `RssCacheDependency` that invalidates a cache key if an RSS (Rich Site Summary) XML Document has changed.

Listing 23-4: Creating an RssCacheDependency class

VB

```
Imports System
Imports System.Web
Imports System.Threading
Imports System.Web.Caching
Imports System.Xml

Public Class RssCacheDependency
        Inherits CacheDependency

    Dim backgroundThread As Timer
    Dim howOften As Integer = 900
    Dim RSS As XmlDocument
    Dim RSSUrl As String

    Public Sub New(ByVal URL As String, ByVal polling As Integer)
        howOften = polling
        RSSUrl = URL
        RSS = RetrieveRSS(RSSUrl)

        If backgroundThread Is Nothing Then
            backgroundThread = New Timer( _
                New TimerCallback(AddressOf CheckDependencyCallback), _
                Me, (howOften * 1000), (howOften * 1000))
        End If
    End Sub

    Function RetrieveRSS(ByVal URL As String) As XmlDocument
        Dim retVal As New XmlDocument
        retVal.Load(URL)
        Return retVal
    End Function

    Public Sub CheckDependencyCallback(ByVal Sender As Object)
        Dim CacheDepends As RssCacheDependency = _
            CType(Sender, RssCacheDependency)
        Dim NewRSS As XmlDocument = RetrieveRSS(RSSUrl)
        If Not NewRSS.OuterXml = RSS.OuterXml Then
            CacheDepends.NotifyDependencyChanged(CacheDepends, EventArgs.Empty)
        End If
        End Sub

    Protected Overrides Sub DependencyDispose()
        backgroundThread = Nothing
        MyBase.DependencyDispose()
    End Sub

    Public ReadOnly Property Document() As XmlDocument
        Get
            Return RSS
        End Get
    End Property
End Class
```

C#

```
using System;
using System.Web;
using System.Threading;
using System.Web.Caching;
using System.Xml;
```

```
public class RssCacheDependency : CacheDependency
{
    Timer backgroundThread;
    int howOften = 900;
    XmlDocument RSS;
    string RSSUrl;

    public RssCacheDependency(string URL, int polling)
    {
        howOften = polling;
        RSSUrl = URL;
        RSS = RetrieveRSS(RSSUrl);

        if (backgroundThread == null)
        {
            backgroundThread = new Timer(
                   new TimerCallback(CheckDependencyCallback),
                   this, (howOften * 1000), (howOften * 1000));
        }
    }

    public XmlDocument RetrieveRSS(string URL)
    {
        XmlDocument retVal = new XmlDocument();
        retVal.Load(URL);
        return retVal;
    }

    public void CheckDependencyCallback(object sender)
    {
        RssCacheDependency CacheDepends = sender as RssCacheDependency;
        XmlDocument NewRSS = RetrieveRSS(RSSUrl);
        if (NewRSS.OuterXml != RSS.OuterXml)
        {
            CacheDepends.NotifyDependencyChanged(CacheDepends, EventArgs.Empty);
        }
    }

    override protected void DependencyDispose()
    {
        backgroundThread = null;
        base.DependencyDispose();
    }

    public XmlDocument Document
    {
        get
        {
            return RSS;
        }
    }
}
```

Create a new Web site and put the `RssCacheDependency` class in a `/Code` folder. Create a `default.aspx` and drag two text boxes, a label, and a button onto the HTML Design view. Execute the Web site and enter an RSS URL for a blog (like mine at `www.hanselman.com/blog/SyndicationService.asmx/GetRss`), and click the button. The program checks the `Cache` object using the URL itself as a key. If the

`XmlDocument` containing RSS doesn't exist in the cache, a new `RssCacheDependency` is created with a 10-minute (600-second) timeout. The `XmlDocument` is then cached, and all future requests within the next 10 minutes to this page retrieve the RSS `XmlDocument` from the cache.

Next, your new `RssCacheDependency` class from Listing 23-4 is illustrated in the following fragment. The `RssCacheDependency` is created and passed into the call to `Cache.Insert`. The `Cache` object handles the lifetime and calling of the methods of the `RssCacheDependency` instance:

VB

```
<%@ Page Language="VB" ValidateRequest="false" %>

<html>
<head runat="server">
    <title>Custom Cache Dependency Example</title>
</head>
<body>
    <form runat="server"> RSS URL:
       <asp:TextBox ID="TextBox1" Runat="server"/>
       <asp:Button ID="Button1" onclick="Button1_Click" Runat="server"
        Text="Get RSS" />
        Cached:<asp:Label ID="Label2" Runat="server"></asp:Label><br />
        RSS:<br />
        <asp:TextBox ID="TextBox2" Runat="server" TextMode="MultiLine"
         Width="800px" Height="300px"></asp:TextBox>
    </form>
</body>
</html>

<script runat="server">
    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim RSSUrl As String = TextBox1.Text
        Label2.Text = "Loaded From Cache"
        If Cache(TextBox1.Text) Is Nothing Then
            Label2.Text = "Loaded Fresh"
            Dim itDepends As New RssCacheDependency(RSSUrl, 600)
            Cache.Insert(RSSUrl, itDepends.Document, itDepends)
        End If
        TextBox2.Text = CType(Cache(TextBox1.Text), _
            System.Xml.XmlDocument).OuterXml
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" ValidateRequest="false" %>
<script runat="server">
    void Button1_Click(object sender, System.EventArgs e)
    {
        string RSSUrl = TextBox1.Text;
        Label2.Text = "Loaded From Cache";
        if (Cache[TextBox1.Text] == null)
        {
            Label2.Text = "Loaded Fresh";
            RssCacheDependency itDepends = new RssCacheDependency(RSSUrl, 600);
            Cache.Insert(RSSUrl, itDepends.Document, itDepends);
        }
```

```
            TextBox2.Text = ((System.Xml.XmlDocument)Cache[TextBox1.Text]).OuterXml;
        }
    </script>
```

The `RssCacheDependency` class creates a `Timer` background thread to poll for changes in the RSS feed. If it detects changes, the `RssCacheDependency` notifies the caching subsystem with the `NotifyDependencyChanged` event. The cached value with that key clears, and the next page view forces a reload of the requested RSS from the specified feed.

Using the SQL Server Cache Dependency

To utilize the SQL Server Cache Dependency feature in ASP.NET, you must perform a one-time setup of your SQL Server database. To set up your SQL Server, use the `aspnet_regsql.exe` tool found at `C:\Windows\MicroSoft.NET\Framework\v2.0.50727/`. This tool makes the necessary modifications to SQL Server so that you can start working with the new SQL cache invalidation features.

Follow these steps when using the new SQL Server Cache Dependency features:

1. Enable your database for SQL Cache Dependency support.
2. Enable a table or tables for SQL Cache Dependency support.
3. Include SQL connection string details in the ASP.NET application's `web.config`.
4. Utilize the SQL Cache Dependency features in one of the following ways:
    - ❑ Programmatically create a `SqlCacheDependency` object in code.
    - ❑ Add a `SqlDependency` attribute to an `OutputCache` directive.
    - ❑ Add a `SqlCacheDependency` instance to the Response object via `Response.AddCacheDependency`.

This section explains all the steps required and the operations available to you.

To start, you need to get at the `aspnet_regsql.exe` tool. Open up the Visual Studio Command Prompt by choosing Start ➪ All Programs ➪ Microsoft Visual Studio 2008 ➪ Visual Studio Tools ➪ Visual Studio Command Prompt from the Windows Start menu. After the prompt launches, type this command:

```
aspnet_regsql.exe -?
```

This code outputs the help command list for this command-line tool, as shown in the following:

```
                        -- SQL CACHE DEPENDENCY OPTIONS --

-d <database>              Database name for use with SQL cache dependency. The
                           database can optionally be specified using the
                           connection string with the -c option instead.
                           (Required)

-ed                        Enable a database for SQL cache dependency.
```

```
-dd                         Disable a database for SQL cache dependency.

-et                         Enable a table for SQL cache dependency. Requires -t
                            option.

-dt                         Disable a table for SQL cache dependency. Requires -t
                            option.

-t <table>                  Name of the table to enable or disable for SQL cache
                            dependency. Requires -et or -dt option.

-lt                         List all tables enabled for SQL cache dependency.
```

The following sections show you how to use some of these commands.

Enabling Databases for SQL Server Cache Invalidation

To use SQL Server cache invalidation with SQL Server 7 or 2000, begin with two steps. The first step enables the appropriate database. In the second step, you enable the tables that you want to work with. You must perform both steps for this process to work. If you want to enable your databases for SQL cache invalidation and you are working on the computer where the SQL Server instance is located, you can use the following construct. If your SQL instance is on another computer, change `localhost` in this example to the name of the remote machine.

```
aspnet_regsql.exe -S localhost -U sa -P password -d Northwind -ed
```

This produces something similar to the following output:

```
Enabling the database for SQL cache dependency.
..
Finished.
```

From this command prompt, you can see that we simply enabled the Northwind database (the sample database that comes with SQL Server) for SQL cache invalidation. The name of the SQL machine was passed in with `-S`, the username with `-U`, the database with `-d`, and most importantly, the command to enable SQL cache invalidation was `-ed`.

Now that you have enabled the database for SQL cache invalidation, you can enable one or more tables contained within the Northwind database.

Enabling Tables for SQL Server Cache Invalidation

You enable more tables by using the following command:

```
aspnet_regsql.exe -S localhost -U sa -P password -d Northwind -t Customers -et

aspnet_regsql.exe -S localhost -U sa -P password -d Northwind -t Products -et
```

You can see that this command is not much different from the one for enabling the database, except for the extra `-t Customers` entry and the use of `-et` to enable the table rather than `-ed` to enable a database. `Customers` is the name of the table that is enabled in this case.

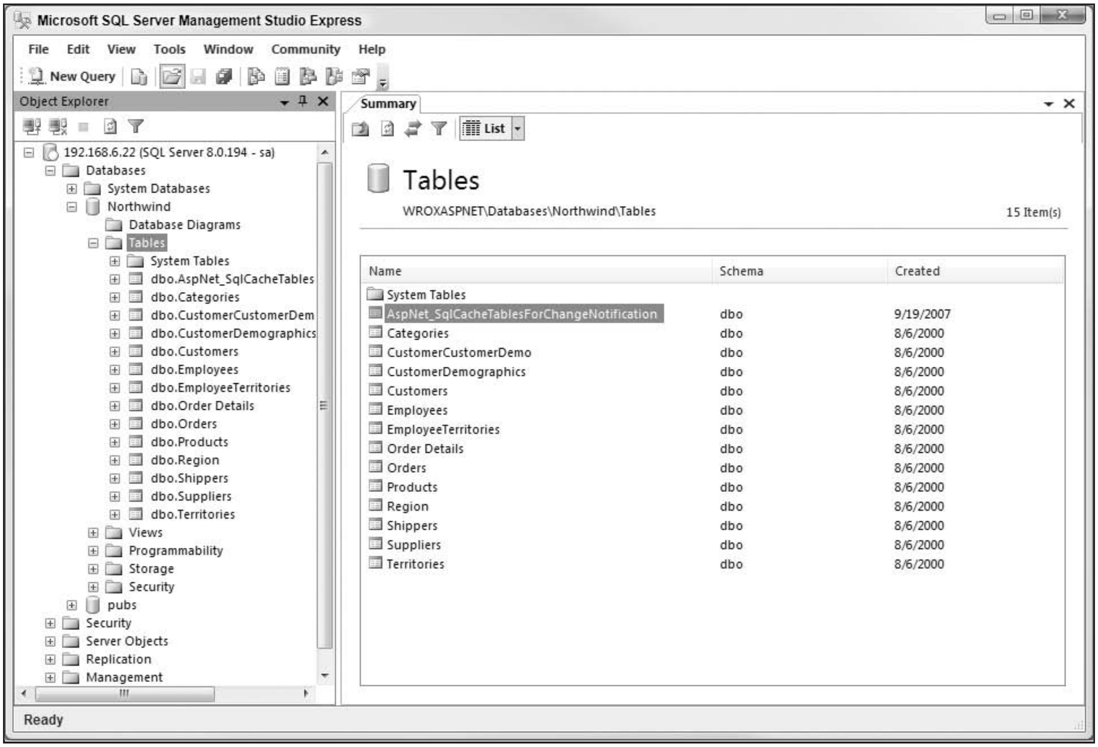
Go ahead and enable both the Customers and Product tables. You run the command once per table. After a table is successfully enabled, you receive the following response:

```
Enabling the table for SQL cache dependency.
.
Finished.
```

After the table is enabled, you can begin using the SQL cache invalidation features. However, before you do, the following section shows you what happens to SQL Server when you enable these features.

Looking at SQL Server 2000

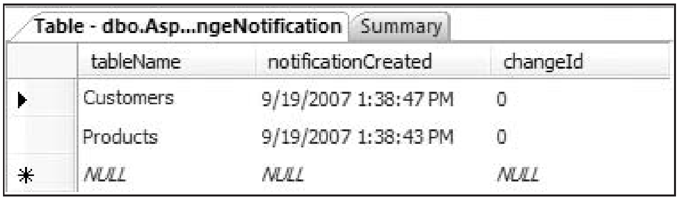
Now that the Northwind database and the Customers and Products tables have all been enabled for SQL cache invalidation, look at what has happened in SQL Server. If you open up the SQL Server Enterprise Manager, you see a new table contained within the Northwind database — `AspNet_SqlCacheTablesForChangeNotification` (whew, that's a long one!). Your screen should look like Figure 23-4. Note that SQL Server 2000 isn't supported on Vista, so this is a screenshot of a *remote* SQL 2000 machine viewed from the SQL Management Studio running on Vista.


Microsoft SQL Server Management Studio Express
File
Edit
View
Tools
Window
Community
Help
New Query
Object Explorer
192.168.6.22 (SQL Server 8.0.194 - sa)
Databases
System Databases
Northwind
Database Diagrams
Tables
System Tables
dbo.AspNet_SqlCacheTables
dbo.Categories
dbo.CustomerCustomerDem
dbo.CustomerDemographics
dbo.Customers
dbo.Employees
dbo.EmployeeTerritories
dbo.Order Details
dbo.Orders
dbo.Products
dbo.Region
dbo.Shippers
dbo.Suppliers
dbo.Territories
Views
Programmability
Storage
Security
pubs
Security
Server Objects
Replication
Management
Summary
List
Tables
WROXASPNET\Databases\Northwind\Tables
15 Item(s)
Ready

Name	Schema	Created
System Tables		
AspNet_SqlCacheTablesForChangeNotification	dbo	9/19/2007
Categories	dbo	8/6/2000
CustomerCustomerDemo	dbo	8/6/2000
CustomerDemographics	dbo	8/6/2000
Customers	dbo	8/6/2000
Employees	dbo	8/6/2000
EmployeeTerritories	dbo	8/6/2000
Order Details	dbo	8/6/2000
Orders	dbo	8/6/2000
Products	dbo	8/6/2000
Region	dbo	8/6/2000
Shippers	dbo	8/6/2000
Suppliers	dbo	8/6/2000
Territories	dbo	8/6/2000

Figure 23-4

At the top of the list of tables in the right-hand pane, you see the `AspNet_SqlCacheTablesForChangeNotification` table. This is the table that ASP.NET uses to learn which tables are being monitored for change notification and also to make note of any changes to the tables being monitored. The table is actually quite simple when you look at the details, as illustrated in Figure 23-5.



	tableName	notificationCreated	changeId
▶	Customers	9/19/2007 1:38:47 PM	0
	Products	9/19/2007 1:38:43 PM	0
*	*NULL*	*NULL*	*NULL*

Figure 23-5

In this figure, you can see three columns in this new table. The first is the `tableName` column. This column simply shows a `String` reference to the names of the tables contained in the same database. Any table named here is enabled for SQL cache invalidation.

The second column, `notificationCreated`, shows the date and time when the table was enabled for SQL cache invalidation. The final column, `changeId`, is used to communicate to ASP.NET any changes to the included tables. ASP.NET monitors this column for changes and, depending on the value, either uses what is stored in memory or makes a new database query.

Looking at the Tables That Are Enabled

Using the `aspnet_regsql.exe` tool, you can see (by using a simple command) which tables are enabled in a particular database. If you are working through the preceding examples, you see that so far you have enabled the Customers and Products tables of the Northwind database. To get a list of the tables that are enabled, use something similar to the following command:

```
aspnet_regsql.exe -S localhost -U sa -P password -d Northwind -lt
```

The `-lt` command produces a simple list of tables enabled for SQL cache invalidation. Inputting this command produces the following results:

```
Listing all tables enabled for SQL cache dependency:
Customers
Products
```

Disabling a Table for SQL Server Cache Invalidation

Now that you know how to enable your SQL Server database for SQL Server cache invalidation, take a look at how you remove the capability for a specific table to be monitored for this process. To remove a table from the SQL Server cache invalidation process, use the `-dt` command.

In the preceding example, using the `-lt` command showed that you have both the Customers and Products tables enabled. Next, you remove the Products table from the process using the following command:

```
aspnet_regsql.exe -S localhost -U sa -P password -d Northwind -t Products -dt
```

You can see that all you do is specify the name of the table using the `-t` command followed by a `-dt` command (disable table). The command line for disabling table caching will again list the tables that are enabled for SQL Server cache invalidation; this time, the Products table is not listed — instead, Customers, the only enabled table, is listed.

Disabling a Database for SQL Server Cache Invalidation

Not only can you pick and choose the tables that you want to remove from the process, but you can also disable the entire database for SQL Server cache invalidation. In order to disable an entire database, you use the `-dd` command (disable database).

> *Note that disabling an entire database for SQL Server cache invalidation also means that every single table contained within this database is also disabled.*

This example shows the Northwind database being disabled on my computer:

```
C:\>aspnet_regsql -S localhost -U sa -P wrox -d Northwind -dd
Disabling the database for SQL cache dependency.
..
Finished.
```

To ensure that the table is no longer enabled for SQL Server cache invalidation, we attempted to list the tables that were enabled for cache invalidation using the `-lt` command. We received the following error:

```
C:\ >aspnet_regsql -S localhost -U sa -P wrox -d Northwind -lt
An error has happened. Details of the exception:
The database is not enabled for SQL cache notification. To enable a database for
SQL cache notification, please use SQLCacheDependencyAdmin.EnableNotifications
method, or the command line tool aspnet_regsql.exe.
```

If you now open the Northwind database in the SQL Server Enterprise Manager, you can see that the `AspNet_SqlCacheTablesForChangeNotification` table has been removed for the database.

SQL Server 2005 Cache Invalidation

As you've seen, standard SQL Server 2000 cache invalidation uses a table-level mechanism using a polling model every few seconds to monitor what tables have changed. SQL Server 2000's technique not only requires preparation of the database, its polling is rather expensive and its caching is quite coarse.

SQL Server 2005 supports a different, more granular series of notification that doesn't require polling. Direct notification of changes is a built-in feature of SQL Server 2005 and is presented via the ADO.NET `SqlCommand`. For example:

```
Protected Sub Page_Load(ByVal sender as Object, ByVal e as System.EventArgs)

    Response.Write("Page created: " + DateTime.Now.ToLongTimeString())
    Dim connStr As String =
ConfigurationManager.ConnectionStrings("AppConnectionString1").ConnectionString
    SqlDependency.Start(connStr)
    Dim connection As New SqlConnection(connStr)
```

```
        Dim command as New SqlCommand("Select * FROM Customers", connection)
        Dim depends as New SqlCacheDependency(command)

        Connection.Open
        GridView1.DataSource = command.ExecuteReader()
        GridView1.DataBind()

        Connection.Close

         "Now, do what you want with the sqlDependency object like:
        Response.AddCacheDependency(depends)

    End Sub
```

SQL Server 2005 supports both programmatic and declarative techniques when caching. Use the string `"CommandNotification"` in the `OutputCache` directive to enable notification-based caching for a page as in this example. You can specify SQL caching options either programmatically or declaratively, but not both. Note that you must first call `System.Data.SqlClient.SqlDependency.Start`, passing in the connection string, to start the SQL notification engine.

```
<%@ OutputCache Duration="3600" VaryByParam="none"
    SqlDependency="CommandNotification"%>
```

Or, if you're using a SqlDataSource control from within your ASP.NET page:

```
<asp:SqlDataSource EnableCaching="true" SqlCacheDependency="CommandNotification"
    CacheDuration="2600" />
```

As data changes within SQL Server 2005, SQL and ADO.NET automatically invalidate data cached on the Web server.

Configuring Your ASP.NET Application

After you enable a database for SQL Server cache invalidation and also enable a couple of tables within this database, the next step is to configure your application for SQL Server cache invalidation.

To configure your application to work with SQL Server cache invalidation, the first step is to make some changes to the `web.config` file. In the `web.config` file, specify that you want to work with the Northwind database, and you want ASP.NET connected to it.

Listing 23-5 shows an example of how you should change your `web.config` file to work with SQL Server cache invalidation. The `pollTime` attribute isn't needed if you're using SQL Server 2005 notification because it uses database events instead of the polling needed for earlier versions.

Listing 23-5: Configuring the web.config file

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">

   <connectionStrings>
      <add name="AppConnectionString1" connectionString="Data Source=localhost;
```

```
         User ID=sa;Password=wrox;Database=Northwind;Persist Security Info=False"
         providerName="System.Data.SqlClient" />
   </connectionStrings>

   <system.web>

      <caching>
         <sqlCacheDependency enabled="true">
            <databases>
               <add name="Northwind" connectionStringName="AppConnectionString1"
                pollTime="500" />
            </databases>
         </sqlCacheDependency>
      </caching>

   </system.web>
</configuration>
```

From this listing, you can see that the first thing established is the connection string to the Northwind database using the `<connectionStrings>` element in the `web.config` file. Note the name of the connection string because it is utilized later in the configuration settings for SQL Server cache invalidation.

The SQL Server cache invalidation is configured using the new `<caching>` element. This element must be nested within the `<system.web>` elements. Because you are working with a SQL Server cache dependency, you must use a `<sqlCacheDependency>` child node. You enable the entire process by using the `enabled = "true"` attribute. After this attribute is enabled, you work with the `<databases>` section. You use the `<add>` element, nested within the `<databases>` nodes, to reference the Northwind database. The following table explains all the attributes of the `<add>` element.

Attribute	**Description**
`Name`	The `name` attribute provides an identifier to the SQL Server database.
`connectionStringName`	The `connectionStringName` attribute specifies the name of the connection. Because the connection string in the preceding example is called `AppConnectionString1`, you use this value for the `connectionStringName` attribute as well.
`pollTime`	The `pollTime` attribute specifies the time interval from one SQL Server poll to the next. The default is .5 seconds or 500 milliseconds (as shown in the example). This is not needed for SQL Server 2005 notification.

Now that the `web.config` file is set up correctly, you can start using SQL Server cache invalidation on your pages. ASP.NET makes a separate SQL Server request on a completely different thread to the `AspNet_SqlCacheTablesForChangeNotification` table to see if the `changeId` number has been incremented. If the number is changed, ASP.NET knows that an underlying change has been made to the SQL Server table and that a new result set should be retrieved. When it checks to see if it should make a SQL Server call, the request to the small `AspNet_SqlCacheTablesForChangeNotification` table has a single result. With SQL Server cache invalidation enabled, this is done so quickly that you really notice the difference.

Testing SQL Server Cache Invalidation

Now that the `web.config` file is set up and ready to go, the next step is to actually apply these new capabilities to a page. Listing 23-6 is an example of a page using the SQL Server cache invalidation process.

Listing 23-6: An ASP.NET page utilizing SQL Server cache invalidation

VB

```
<%@ Page Language="VB" %>
<%@ OutputCache Duration="3600" VaryByParam="none"
    SqlDependency="Northwind:Customers"%>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "Page created at " & DateTime.Now.ToShortTimeString ()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Sql Cache Invalidation</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Label ID="Label1" Runat="server"></asp:Label><br />
        <br />
        <asp:GridView ID="GridView1" Runat="server" DataSourceID="SqlDataSource1">
        </asp:GridView>
        <asp:SqlDataSource ID="SqlDataSource1" Runat="server"
         SelectCommand="Select * From Customers"
         ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>"
         ProviderName="<%$ ConnectionStrings:AppConnectionString1.providername %>">
        </asp:SqlDataSource>
    </form>
</body>
</html>
```

C#

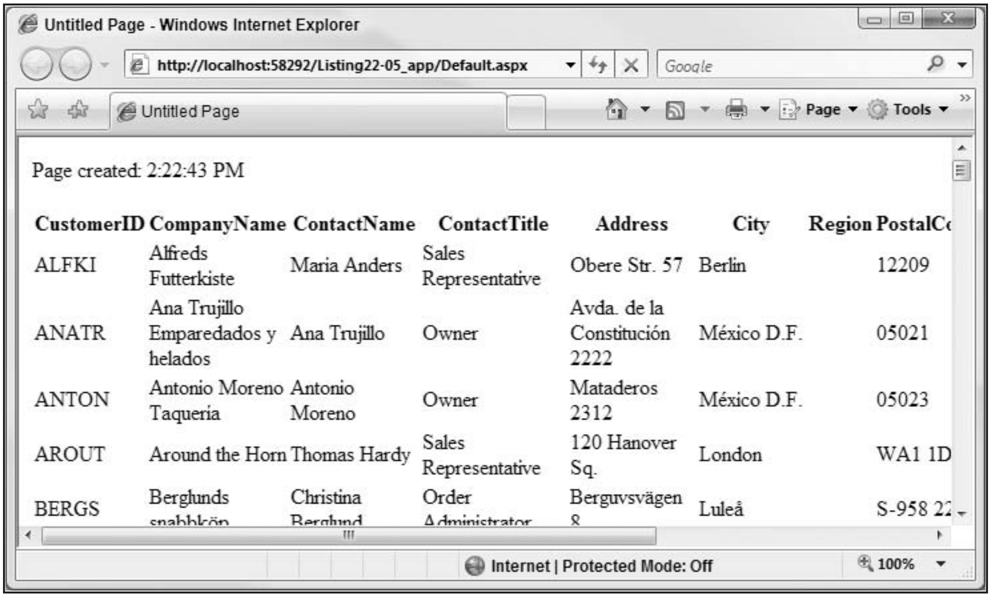
```
<%@ Page Language="C#" %>
<%@ OutputCache Duration="3600" VaryByParam="none"
    SqlDependency="Northwind:Customers"%>

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
        Label1.Text = "Page created at " + DateTime.Now.ToShortTimeString();
    }
</script>
```

The first and most important part of this page is the `OuputCache` page directive that is specified at the top of the file. Typically, the `OutputCache` directive specifies how long the page output is held in the cache using the `Duration` attribute. Next is the `VaryByParam` attribute. The new addition is the `SqlDependency` attribute. This enables a particular page to use SQL Server cache invalidation. The following line shows the format of the value for the `SqlDependency` attribute:

```
SqlDependency="database:table"
```

The value of `Northwind:Customers` specifies that you want the SQL Server cache invalidation enabled for the Customers table within the Northwind database. The `Duration` attribute of the `OutputCache` directive shows you that, typically, the output of this page is stored in the cache for a long time — but this cache is invalidated immediately if the Customers table has any underlying changes made to the data that it contains.

A change to any of the cells in the Customers table of the Northwind database invalidates the cache, and a new cache is generated from the result, which now contains a new SQL Server database request. Figure 23-6 shows an example of the page generated the first time it is run.

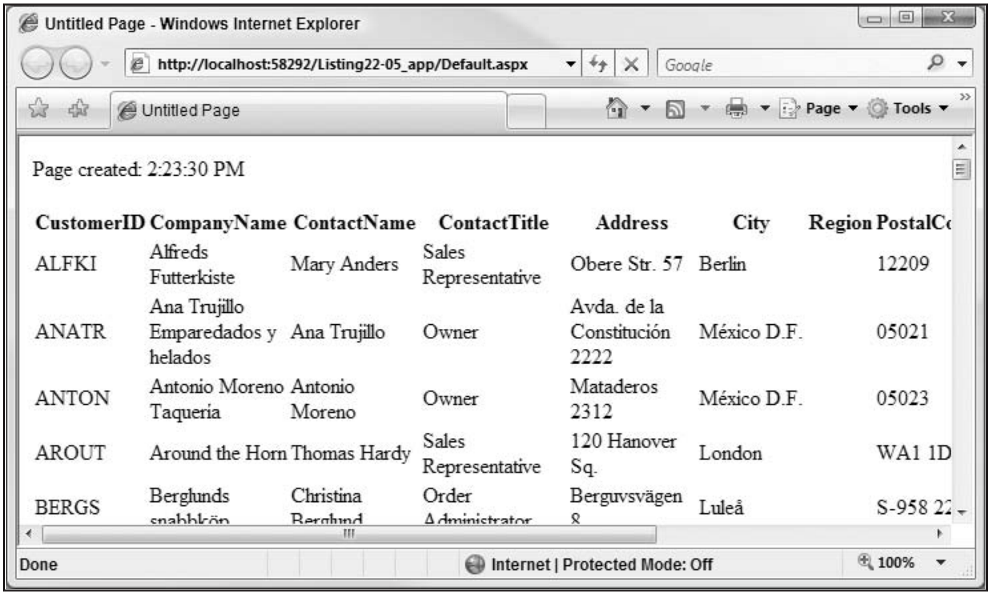

Untitled Page - Windows Internet Explorer
http://localhost:58292/Listing22-05_app/Default.aspx
Google
Untitled Page
Page
Tools
Internet | Protected Mode: Off
100%

Page created: 2:22:43 PM

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalC
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1D
BERGS	Berglunds	Christina	Order	Berguvsvägen	Luleå		S-958 22

Figure 23-6

From this figure, you can see the contents of the customer with the `CustomerID` of `ALFKI`. For this entry, go to SQL Server and change the value of the `ContactName` from `Maria Anders` to `Mary Anders`. If we weren't using SQL Server cache invalidation, this change would have done nothing to the output cache. The original page output in the cache would still be present and the end user would still see the `Maria Anders` entry for the duration specified in the page's `OutputCache` directive. But because we're using SQL Server cache invalidation, after the underlying information in the table is changed, the output cache is invalidated, a new result set is retrieved, and the new result set is cached. When a change has been made, you see the results as shown in Figure 23-7.


Untitled Page - Windows Internet Explorer
http://localhost:58292/Listing22-05_app/Default.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

Page created: 2:23:30 PM

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalC
ALFKI	Alfreds Futterkiste	Mary Anders	Sales Representative	Obere Str. 57	Berlin		12209
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1D
BERGS	Berglunds snabbkön	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22

Figure 23-7

Notice also that the text ''Page created at'' includes an updated time indicating when this page was rendered. Need to stop working so late, eh?

Adding More Than One Table to a Page

The preceding example shows how to use SQL Server cache invalidation for a single table on the ASP.NET page. What do you do if your page is working with two or more tables?

To add more than one table, you use the `OutputCache` directive shown here:

```
SqlDependency="database:table;database:table"
```

From this example, you can see that the value of the `SqlDependency` attribute separates the databases and tables with a semicolon. If you want to work with both the Customers table and the Products table of the Northwind database, you construct the value of the `SqlDependency` attribute as follows:

```
SqlDependency="Northwind:Customers;Northwind:Products"
```

Attaching SQL Server Cache Dependencies to the Request Object

In addition to changing settings in the `OutputCache` directive to activate SQL Server cache invalidation, you can also set the SQL Server cache invalidation programmatically. To do so, use the `SqlCacheDependency` class, which is illustrated in Listing 23-7.

Listing 23-7: Working with SQL Server cache invalidation programmatically

VB

```
Dim myDependency As SqlCacheDependency = _
   New SqlCacheDependency("Northwind", "Customers")
Response.AddCacheDependency(myDependency)
Response.Cache.SetValidUntilExpires(true)
Response.Cache.SetExpires(DateTime.Now.AddMinutes(60))
Response.Cache.SetCacheability(HttpCacheability.Public)
```

C#

```
SqlCacheDependency myDependency = new SqlCacheDependency("Northwind", "Customers");
Response.AddCacheDependency(myDependency);
Response.Cache.SetValidUntilExpires(true);
Response.Cache.SetExpires(DateTime.Now.AddMinutes(60));
Response.Cache.SetCacheability(HttpCacheability.Public);
```

You first create an instance of the `SqlCacheDependency` object, assigning it the value of the database and the table at the same time. The `SqlCacheDependency` class takes the following parameters:

```
SqlCacheDependency(databaseEntryName As String, tablename As String)
```

You use this parameter construction if you are working with SQL Server 7.0 or with SQL Server 2000. If you are working with SQL Server 2005, you use the following construction:

```
SqlCacheDependency(sqlCmd As System.Data.SqlClient.SqlCommand)
```

After the `SqlCacheDependency` class is in place, you add the dependency to the `Cache` object and set some of the properties of the `Cache` object as well. You can do this either programmatically or through the `OutputCache` directive.

Attaching SQL Server Cache Dependencies to the Cache Object

In addition to attaching SQL Server cache dependencies to the `Request` object, you can attach them to the `Cache` object for data that can be cached much longer. The `Cache` object is contained within the `System.Web.Caching` namespace, and it enables you to work programmatically with the caching of any type of objects. Listing 23-8 shows a page that utilizes the `Cache` object with the `SqlDependency` object.

Listing 23-8: Using the Cache object with the SqlDependency object

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data"%>
<%@ Import Namespace="System.Data.SqlClient"%>
```

```
<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim myCustomers As DataSet
        myCustomers = CType(Cache("firmCustomers"), DataSet)

        If myCustomers Is Nothing Then
           Dim conn As SqlConnection = _
            New SqlConnection( _
 ConfigurationManager.ConnectionStrings("AppConnectionString1").ConnectionString)
           Dim da As SqlDataAdapter = _
            New SqlDataAdapter("Select * From Customers", conn)

           myCustomers = New DataSet
           da.Fill(myCustomers)

           Dim myDependency As SqlCacheDependency = _
              New SqlCacheDependency("Northwind", "Customers")
           Cache.Insert("firmCustomers", myCustomers, myDependency)

           Label1.Text = "Produced from database."
        Else
           Label1.Text = "Produced from Cache object."
        End If

        GridView1.DataSource = myCustomers
        GridView1.DataBind()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Sql Cache Invalidation</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Label ID="Label1" Runat="server"></asp:Label><br />
        <br />
        <asp:GridView ID="GridView1" Runat="server"></asp:GridView>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
       DataSet myCustomers;
       myCustomers = (DataSet)Cache["firmCustomers"];

       if (myCustomers == null)
       {
          SqlConnection conn = new
```

```
              SqlConnection(
ConfigurationManager.ConnectionStrings["AppConnectionString1"].ConnectionString);
            SqlDataAdapter da = new
               SqlDataAdapter("Select * from Customers", conn);

            myCustomers = new DataSet();
            da.Fill(myCustomers);

            SqlCacheDependency myDependency = new
               SqlCacheDependency("Northwind", "Customers");
            Cache.Insert("firmCustomers", myCustomers, myDependency);

            Label1.Text = "Produced from database.";
          }
          else
          {
            Label1.Text = "Produced from Cache object.";
          }

          GridView1.DataSource = myCustomers;
          GridView1.DataBind();
  }
  </script>
```

In this example, the `SqlCacheDependency` class associated itself to the Customers table in the Northwind database as before. This time, however, you use the `Cache` object to insert the retrieved dataset along with a reference to the `SqlCacheDependency` object. The `Insert` method of the `Cache` class is constructed as follows:

```
Cache.Insert(key As String, value As Object,
   dependencies As System.Web.Caching.CacheDependency)
```

You can also insert more information about the dependency using the following construct:

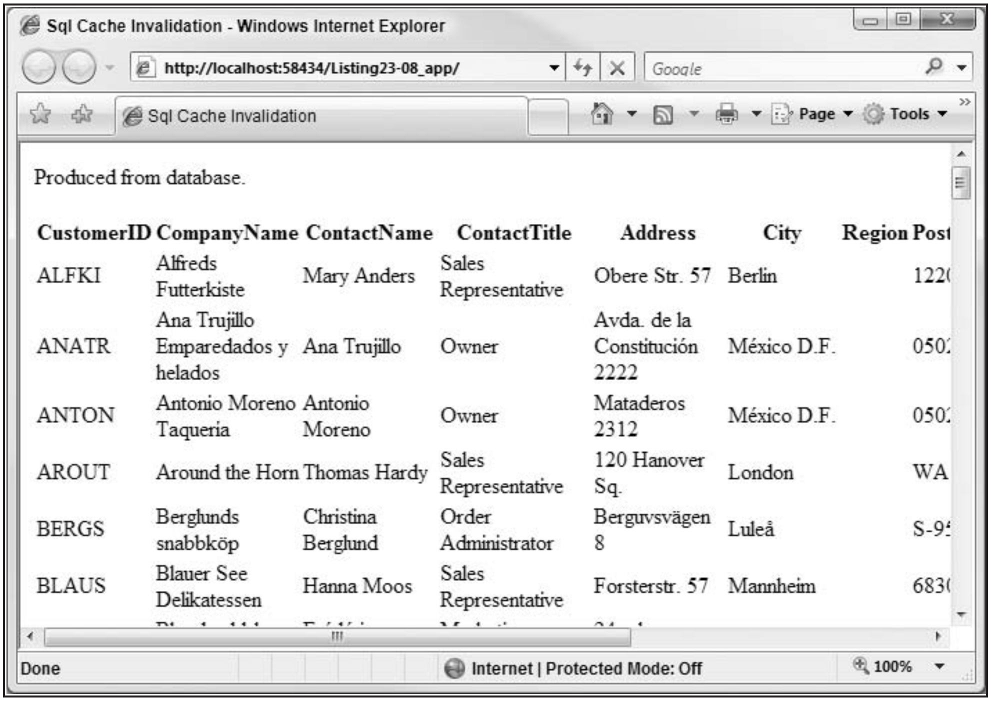
```
Cache.Insert(key As String, value As Object,
   dependencies As System.Web.Caching.CacheDependency
   absoluteExpiration As Date, slidingExpiration As System.TimeSpan)
```

And finally:

```
Cache.Insert(key As String, value As Object,
   dependencies As System.Web.Caching.CacheDependency
   absoluteExpiration As Date, slidingExpiration As System.TimeSpan)
   priority As System.Web.Caching.CacheItemPriority,
   onRemoveCallback As System.Web.Caching.CacheItemRemovedCallback)
```

The SQL Server cache dependency created comes into action and does the same polling as it would have done otherwise. If any of the data in the Customers table has changed, the `SqlCacheDependency` class invalidates what is stored in the cache. When the next request is made, the `Cache("firmCustomers")` is found to be empty and a new request is made to SQL Server. The `Cache` object again repopulates the cache with the new results generated.

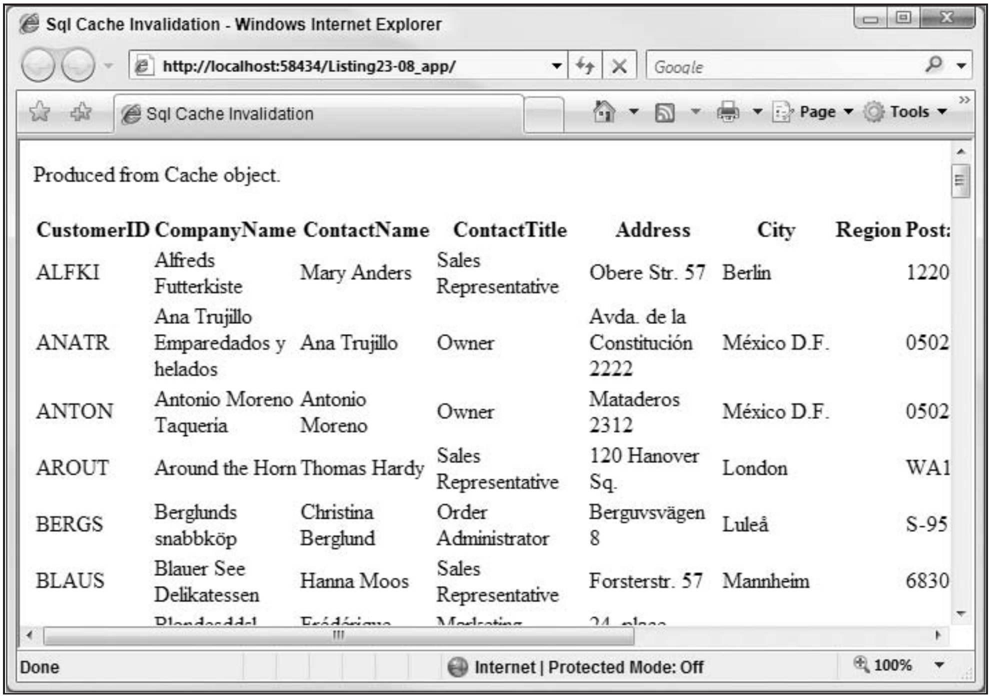
When the ASP.NET page from Listing 23-8 is called for the first time, the results generated are shown in Figure 23-8.


Sql Cache Invalidation - Windows Internet Explorer
http://localhost:58434/Listing23-08_app/
Google
Sql Cache Invalidation
Page
Tools
Done
Internet | Protected Mode: Off
100%

Produced from database.

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	Post
ALFKI	Alfreds Futterkiste	Mary Anders	Sales Representative	Obere Str. 57	Berlin		122
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		050
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	México D.F.		050
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-9
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		683

Figure 23-8


Sql Cache Invalidation - Windows Internet Explorer
http://localhost:58434/Listing23-08_app/
Google
Sql Cache Invalidation
Page
Tools
Done
Internet | Protected Mode: Off
100%

Produced from Cache object.

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	Post
ALFKI	Alfreds Futterkiste	Mary Anders	Sales Representative	Obere Str. 57	Berlin		1220
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		0502
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	México D.F.		0502
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-95
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		6830

Figure 23-9

Because this is the first time that the page is generated, nothing is in the cache. The `Cache` object is, therefore, placed in the result set along with the association to the SQL Server cache dependency. Figure 23-9 shows the result for the second request. Notice that the HTML table is identical because it was generated from the identical `DataSet`, but the first line of the page has changed to indicate that this output was produced from cache.

On the second request, the dataset is already contained within the cache; therefore, it is retrievable. You aren't required to hit SQL Server to get the full results again. If any of the information has changed within SQL Server itself, however, the `Cache` object returns nothing; a new result set is retrieved.

Summary

SQL Server cache invalidation is an outstanding feature of ASP.NET that enables you to invalidate items stored in the cache when underlying changes occur to the data in the tables being monitored. Post-Cache Substitution fills in an important gap in ASP.NET's technology, enabling you to have both the best highly dynamic content and a high-performance Web site with caching.

When you are monitoring changes to the database, you can configure these procedures easily in the `web.config` file, or you can work programmatically with cache invalidation directly in your code. These changes are possible because the `CacheDependency` object has been unsealed. You can now inherit from this object and create your own cache dependencies. The SQL Server cache invalidation process is the first example of this capability.

24

Debugging and Error Handling

Your code always runs exactly as you wrote it, and you will *never* get it right the first time. So, expect to spend about 30 percent of your time debugging and, to be a successful debugger, learn to use the available tools effectively. Visual Studio has upped the ante, giving you a host of new features that greatly improve your debugging experience. So many of these new features, however, can be overwhelming at first. This chapter breaks down all the techniques available to you, one at a time, while presenting a holistic view of Visual Studio, the Common Language Runtime (CLR), and the Base Class Library (BCL).

> *Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?*
>
> ***— Brian Kernighan***

Additionally, because debugging is more than stepping through code, this chapter discusses efficient error and exception handling, tracing and logging, and cross-language (C#, Visual Basic, client-side JavaScript, XSLT, and SQL Stored Procedure) debugging.

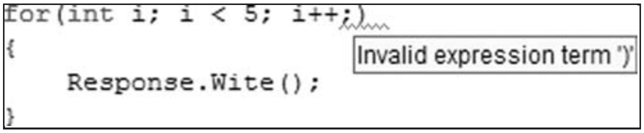
Design-Time Support

Visual Studio has always done a good job of warning you of potential errors at design time. *Syntax notifications* or *squiggles* underline code that won't compile or that might cause an error before you have compiled the project. A *new error notification* pops up when an exception occurs during a debugging session and recommends a course of action that prevents the exception. At every step, Visual Studio tries to be smarter, anticipating your needs and catching common mistakes.

Rico Mariani, the Chief Architect of Visual Studio, has used the term ''The Pit of Success'' to describe the experience Microsoft wants you to have with Visual Studio. When Microsoft designed these new features, they wanted the customer to simply fall into winning practices. The company tried to achieve this by making it more difficult for you to write buggy code or make common mistakes. Microsoft's developers put a great deal of thought into building APIs that point you in the right direction.

Syntax Notifications

Both the Visual Basic and C# editors show squiggles and tooltips for many syntax errors well before compilation, as illustrated in Figure 24-1.



```
for(int i; i < 5; i++;)
{
    Response.Wite();
}
```

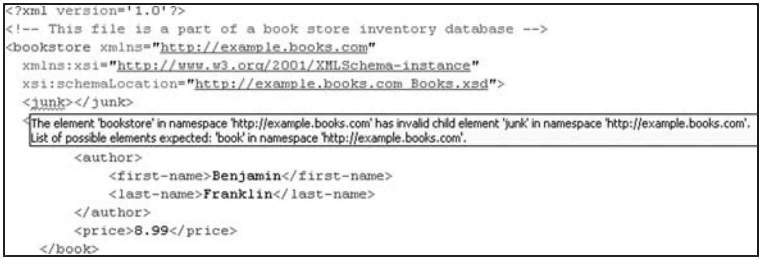
Invalid expression term ')'

Figure 24-1

Syntax notifications aren't just for CLR programming languages; Visual Studio has also greatly improved the XML Editor since its original release with enhancements like the following:

- Full XML 1.0 syntax checking
- Support for XSD and DTD validation and IntelliSense
- Support for XSLT 1.0 syntax checking

Figure 24-2 shows a detailed tooltip indicating that the element `<junk>` doesn't have any business being in the `web.config` file. The editor knows this because of a combination of the XSD validation support in the XML Editor and the addition of schemas for configuration files such as `web.config`. This is a welcome change for anyone who, when manually editing a `web.config` file, has wondered if he guessed the right elements. See the Chapter 10 for more details on these XML-related features.



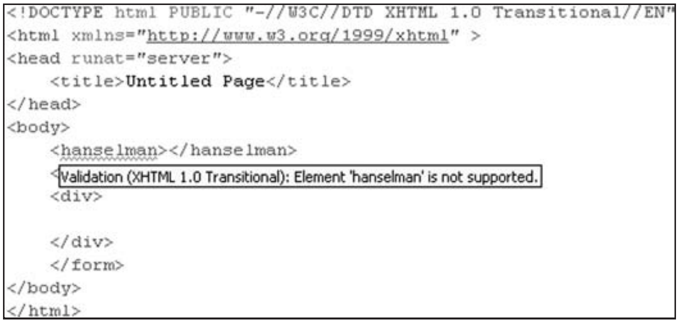
```
<?xml version='1.0'?>
<!-- This file is a part of a book store inventory database -->
<bookstore xmlns="http://example.books.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://example.books.com Books.xsd">
  <junk></junk>
  <
        <author>
            <first-name>Benjamin</first-name>
            <last-name>Franklin</last-name>
        </author>
        <price>8.99</price>
    </book>
```

The element 'bookstore' in namespace 'http://example.books.com' has invalid child element 'junk' in namespace 'http://example.books.com'. List of possible elements expected: 'book' in namespace 'http://example.books.com'.

Figure 24-2

The ASPX/HTML Editor benefits from these improvements as well; for example, Figure 24-3 shows a warning that the `<hanselman/>` element is not available in the active schema. Code that appears in `<script runat="Server"/>` blocks in ASP.NET pages is also parsed and marked with squiggles. This makes including code in your pages considerably easier. Notice also that the ASP.NET page in Figure 24-3 has an XHTML `DOCTYPE` declaration on the first line, and the HTML element has a default XHTML namespace. This HTML page is treated as XML because XHTML has been targeted.

XHTML is the HTML vocabulary of markup expressed with all the syntax rules of XML. For example, in HTML you could create a `<br>` tag and never close it. In XHTML you use the closing tag `<br />`. XHTML documents look exactly like HTML documents because they *are*, in fact, expressing the same semantics. Because XHTML documents are XML, they require a namespace on their root element and should have a `DOCTYPE` as well.

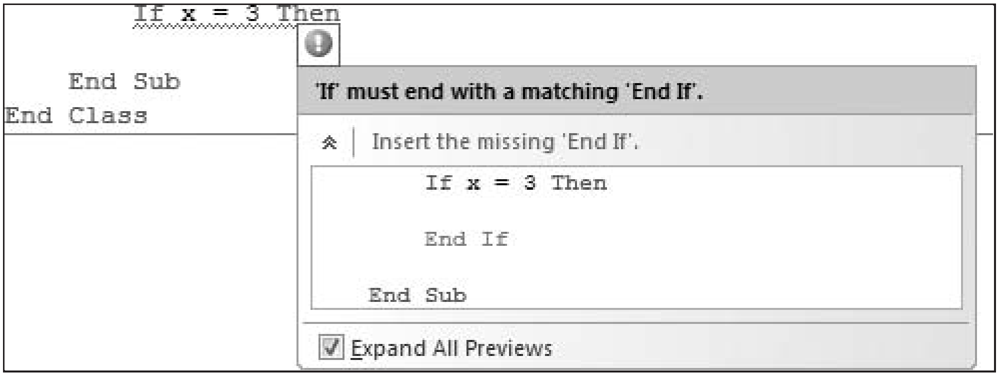


```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <hanselman></hanselman>
    <Validation (XHTML 1.0 Transitional): Element 'hanselman' is not supported.
    <div>

    </div>
    </form>
</body>
</html>
```

Figure 24-3

To add a `hanselman` element, you must put it in its own namespace and add a namespace declaration in the root HTML element.

The Visual Basic Editor takes assistance to the next level with a *smart tag* like the pulldown/button that appears when you hover your mouse over a squiggle. A very nicely rendered modeless window appears with your code in a box along with some suggested changes to make your code compile. Figure 24-4 shows a recommendation to insert a missing `End If`; making the correction is simple — just click `Insert the missing 'End If'`.



```
        If x = 3 Then

    End Sub
End Class
```

'If' must end with a matching 'End If'.

Insert the missing 'End If'.

```
        If x = 3 Then

        End If

    End Sub
```

Figure 24-4

All these design-time features exist to help you ensure better code while it's being written, before it's been compiled and run. Two related features help you run arbitrary code within the development environment as well as organize the tasks still to be performed.

Immediate and Command Window

The Immediate Window lets you run arbitrary bits of code in Design mode without compiling your application. You can evaluate code at design time or while you're debugging. It can be a great way to test a line of code or a static method quickly. The Immediate mode of this window is used primarily for debugging.

Access the Immediate Window from Debug ➪ Windows ➪ Immediate. To evaluate a variable or run a method, simply click in the Immediate Window and type a question mark (**?**) followed by the expression, variable, or method you want to evaluate.

The immediate window can also be switched into the Command Window by prefacing commands with a greater-than sign (>). When you enter a greater-than sign in the Immediate/Command Window, an IntelliSense drop-down appears exposing the complete Visual Studio object model as well as any macros that you may have recorded. Command mode of this window is used for executing Visual Studio Commands without using the menus. You can also execute commands that may not have a menu item.

If you type `>alias` into the Command window, you receive a complete list of all current aliases and their definitions. Some useful command aliases include the following:

- ❑ `>Log filename /overwrite /on|off`: The `Log` command starts logging all output from the command window to a file. If no filename is included for logging, go to `cmdline.log`. This is one of the more useful and least-used features of the debugger, and reason enough to learn a few things about the immediate/Command Window.
- ❑ `>Shell args /command /output /dir:folder`: The `Shell` command allows you to launch executable programs from within the Visual Studio Command window such as utilities, command shells, batch files, and so on.

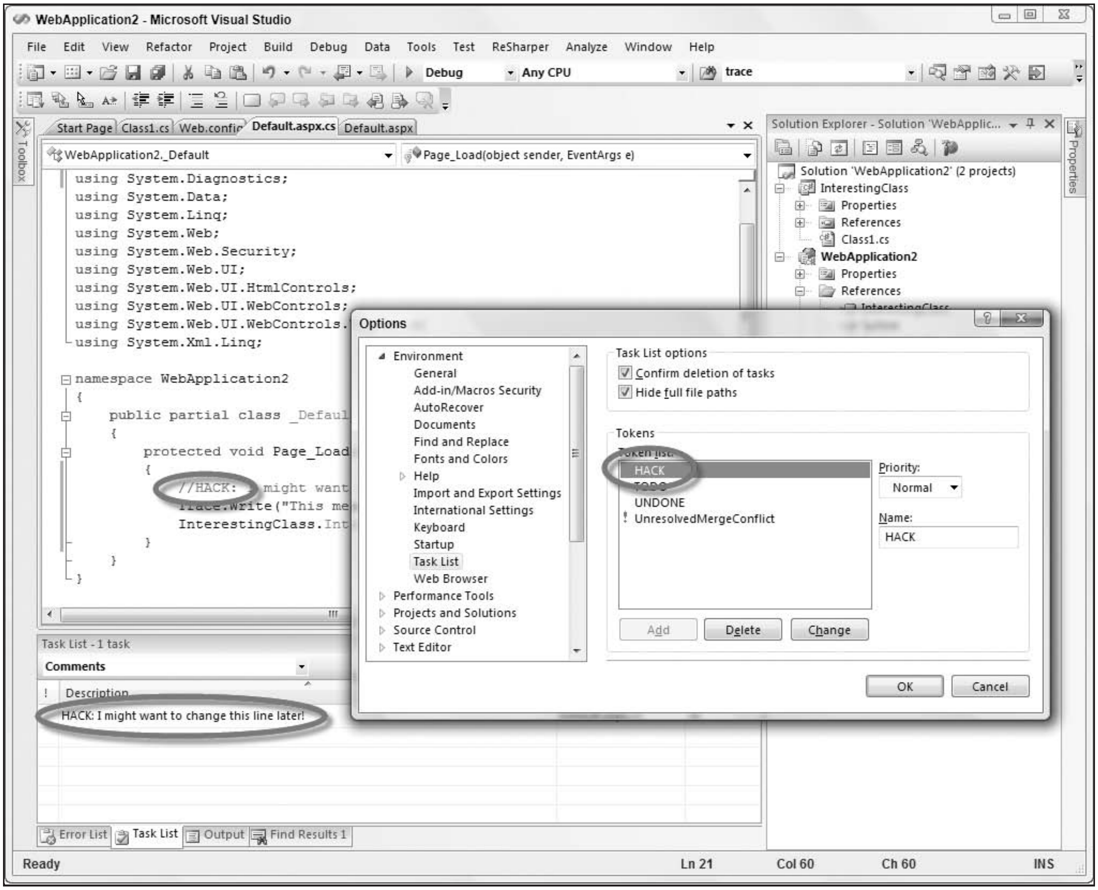
Task List

The Task list in Visual Studio is more useful than you might think. People who haven't given it much attention are missing out on a great feature. The Task list supports two views: User Tasks and Comments.

User Tasks view enables you to add and modify tasks, which can include anything from ''Remember to Test'' to ''Buy Milk.'' These tasks are stored in the .suo (solution user options) that is a parallel partner to the .sln files.

The Comments view shows text from the comments in your code where those lines are prefixed with a specific token. Visual Studio comes configured to look for the `TODO:` token, but you can add your own in Tools ➪ Options ➪ Environment ➪ Task List.

In Figure 24-5, the comment token `HACK` has been added in the Options dialog box. A comment appears in the source with `HACK:` preceding it, so that comment line automatically appears in the Task List in the docked window at the bottom of Visual Studio. The three circles in Figure 24-5 illustrate the connection between the word `HACK` added to the Options dialog box and its subsequent appearance in the source code and Task List. You and your team can add as many of these tokens as you'd like.



```
using System.Diagnostics;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.
using System.Xml.Linq;

namespace WebApplication2
{
    public partial class _Defaul
    {
        protected void Page_Load
        {
            //HACK:  might want
            Trace.Write("This me
            InterestingClass.Int
        }
    }
}
```

Options — Environment > Task List

Task List options

- ☑ Confirm deletion of tasks
- ☑ Hide full file paths

Tokens

Token list:

- HACK
- TODO
- UNDONE
- UnresolvedMergeConflict

Priority: Normal

Name: HACK

Task List - 1 task

HACK: I might want to change this line later!

WebApplication2 - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Data Tools Test ReSharper Analyze Window Help
Debug
Any CPU
trace
Start Page
Class1.cs
Web.config
Default.aspx.cs
Default.aspx
WebApplication2._Default
Page_Load(object sender, EventArgs e)
Solution Explorer - Solution 'WebApplic...
Solution 'WebApplication2' (2 projects)
InterestingClass
Properties
References
Class1.cs
WebApplication2
Properties
References
Environment
General
Add-in/Macros Security
AutoRecover
Documents
Find and Replace
Fonts and Colors
Help
Import and Export Settings
International Settings
Keyboard
Startup
Task List
Web Browser
Performance Tools
Projects and Solutions
Source Control
Text Editor
Add
Delete
Change
OK
Cancel
Comments
Description
Error List
Task List
Output
Find Results 1
Ready
Ln 21
Col 60
Ch 60
INS
Toolbox
Properties

Figure 24-5

Tracing

Tracing is a way to monitor the execution of your ASP.NET application. You can record exception details and program flow in a way that doesn't affect the program's output. In classic ASP, tracing and debugging facilities were nearly nonexistent, forcing developers to use "got here" debugging in the form of many `Response.Write` statements that litter the resulting HTML with informal trace statements announcing to the programmer that the program "got here" and "got there" with each new line executed. This kind of intrusive tracing was very inconvenient to clean up and many programmers ended up creating their own informal trace libraries to get around these classic ASP limitations.

In ASP.NET, there is rich support for tracing. The destination for trace output can be configured with TraceListeners like the `EventLogTraceListener`. Configuration of TraceListeners is covered later in this section. ASP.NET also includes a number of small improvements to tracing over ASP.NET 1.*x*, including trace forwarding between the ASP.NET page-specific `Trace` class and standard Base Class Library's (BCL) `System.Diagnostics.Trace` used by non–Web developers. Additionally, the resolution of the timing output by ASP.NET tracing has increased precision — from 6 digits to 18 digits for highly accurate profiling between ASP.NET 1.1 and ASP.NET 2.0/3.5.

System.Diagnostics.Trace and ASP.NET's Page.Trace

There are multiple things named *Trace* in the whole of the .NET Framework, so it may appear that tracing isn't unified between Web and non-Web applications. Don't be confused because there is a class called `System.Diagnostics.Trace` and there is also a public property on `System.Web.UI.Page` called `Trace`. The `Trace` property on the `Page` class gives you access to the `System.Web.TraceContext` and the ASP.NET-specific tracing mechanism. The `TraceContext` class collects all the details and timing of a Web request. It contains a number of methods, but the one you'll use the most is `Write`. It also includes `Warn`, which simply calls `Write()`, and also ensures that the output generated by `Warn` is colored red.

If you're writing an ASP.NET application that has no supporting components or other assemblies that may be used in a non-Web context, you can usually get a great deal of utility using only the ASP.NET `TraceContext`. However, ASP.NET support tracing is different from the rest of the base class library's tracing. You'll explore ASP.NET's tracing facilities first, and then learn how to bridge the gap and see some new features that make debugging even easier.

Page-Level Tracing

ASP.NET tracing can be enabled on a page-by-page basis by adding `Trace="true"` to the Page directive in any ASP.NET page:

```
<%@ Page Language="C#" Inherits="System.Web.UI.Page" Trace="true" %>
```

Additionally, you can add the `TraceMode` attribute that sets `SortByCategory` or the default, `SortByTime`. You might include a number of categories, one per subsystem, and use `SortByCategory` to group them, or you might use `SortByTime` to see the methods that take up the most CPU time for your application. You can enable tracing programmatically as well, using the `Trace.IsEnabled` property. The capability to enable tracing programmatically means you can enable tracing via a querystring, cookie, or IP address; it's up to you.

Application Tracing

Alternatively, you can enable tracing for the entire application by adding tracing settings in `web.config`. In the following example, `pageOutput="false"` and `requestLimit="20"` are used, so trace information is stored for 20 requests, but not displayed on the page:

```
<configuration>
   <system.web>
       <trace enabled="true" pageOutput="false" requestLimit="20"
           traceMode="SortByTime" localOnly="true" />
   </system.web>
</configuration>
```

The page-level settings take precedence over settings in `web.config`, so if `enabled="false"` is set in `web.config` but `trace="true"` is set on the page, tracing occurs.

Viewing Trace Data

Tracing can be viewed for multiple page requests at the application level by requesting a special page (of sorts) called `trace.axd`. Note that `trace.axd` doesn't actually exist; it is actually provided by `System.Web.Handlers.TraceHandler`, a special `IHttpHandler` to which `trace.axd` is bound. When ASP.NET detects an HTTP Request for `trace.axd`, that request is handled by the `TraceHandler` rather than by a page.

Create a Web site and a page, and in the `Page_Load` event, call `Trace.Write()`. Enable tracing in the `web.config` as shown in Listing 24-1.

Listing 24-1: Tracing using Page.Trace

Web.config

```
<configuration>
   <system.web>
       <trace enabled="true" pageOutput="true" />
   </system.web>
</configuration>
```

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   Handles Me.Load 'All on one line!
       Trace.Write("This message is from the START OF the Page_Load method!")
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
   Trace.Write("This message is from the START of the Page_Load method!");
}
```

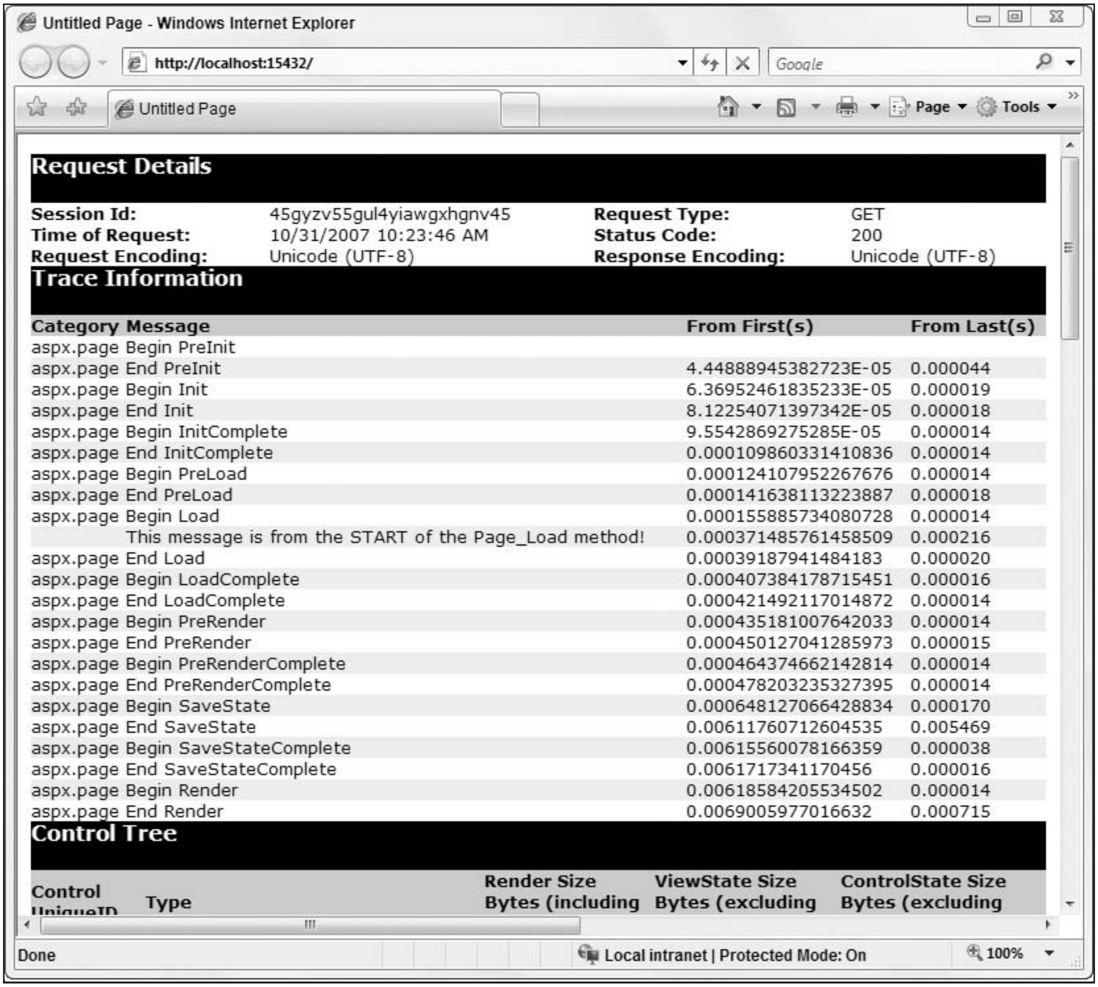
Hit the page in the browser a few times and notice that, although this page doesn't create any HTML to speak of, a great deal of trace information is presented in the browser, as shown in Figure 24-6, because the setting is `pageOutput="true"`.

The message from `Trace.Write` appears after Begin Load and before End Load — it's right in the middle of the `Page_Load` method where you put it. The page was automatically JIT-compiled as you ran it, and that initial performance hit is over. Now that it's been compiled into native code, a subsequent run of this same page, performed by clicking Refresh in the browser, took only 0.000167 seconds on my laptop because the page had already compiled. It's very easy and very useful to collect this kind of very valuable performance timing data between Trace statements.

Incidentally, this simple page is more than 100 times faster than when the first edition of this book was written in 2004. Both computers and the JITter continue to improve!

Eleven different sections of tracing information provide a great deal of detail and specific insight into the ASP.NET page-rendering process, as described in the following table.

Section	Description
Request Details	Includes the ASP.NET Session ID, the character encoding of the request and response, and the HTTP conversation's returned status code. Be aware of the request and response encoding, especially if you're using any non-Latin character sets. If you're returning languages other than English, you'll want your encoding to be UTF-8. Fortunately that is the default.
Trace Information	Includes all the `Trace.Write` methods called during the lifetime of the HTTP request and a great deal of information about timing. This is probably the most useful section for debugging. The timing information located here is valuable when profiling and searching for methods in your application that take too long to execute.
Control Tree	Presents an HTML representation of the ASP.NET Control Tree. Shows each control's unique ID, runtime type, the number of bytes it took to be rendered, and the bytes it requires in ViewState and ControlState. Don't undervalue the usefulness of these two sections, particularly of the three columns showing the weight of each control. The weight of the control indicates the number of bytes occupied in ViewState and/or ControlState by that particular Control. Be aware of the number of bytes that each of your controls uses, especially if you write your own custom controls, as you want your controls to return as few bytes as possible to keep overall page weight down.
Session State	Lists all the keys for a particular user's session, their types, and their values. Shows only the current user's Session State.
Application State	Lists all the keys in the current application's Application object and their types and values.
Request Cookies	Lists all the cookies passed in during the page's request.
Response Cookies	Lists all the cookies that were passed back during the page's response.
Headers Collection	Shows all the headers that might be passed in during the request from the browser, including Accept-Encoding, indicating whether the browser supports compressed HTTP responses; Accept-Languages, a list of ISO language codes that indicate the order of the user's language preferences; and User-Agent, the identifying string for the user's browser. The string also contains information about the user's operating system and the version or versions of the .NET Framework he is running (on IE).
Form Collection	Displays a complete dump of the Form collection and all its keys and values.
Querystring Collection	Displays a dump of the Querystring collection and all its contained keys and values.
Server Variables	A complete dump of name-value pairs of everything that the Web server knows about the application and the requesting browser.


Untitled Page - Windows Internet Explorer
http://localhost:15432/
Google
Untitled Page
Page
Tools
Done
Local intranet | Protected Mode: On
100%

Request Details

Session Id:	45gyzv55gul4yiawgxhgnv45	**Request Type:**	GET
Time of Request:	10/31/2007 10:23:46 AM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	4.44888945382723E-05	0.000044
aspx.page	Begin Init	6.36952461835233E-05	0.000019
aspx.page	End Init	8.12254071397342E-05	0.000018
aspx.page	Begin InitComplete	9.5542869275285E-05	0.000014
aspx.page	End InitComplete	0.000109860331410836	0.000014
aspx.page	Begin PreLoad	0.000124107952267676	0.000014
aspx.page	End PreLoad	0.000141638113223887	0.000018
aspx.page	Begin Load	0.000155885734080728	0.000014
	This message is from the START of the Page_Load method!	0.000371485761458509	0.000216
aspx.page	End Load	0.00039187941484183	0.000020
aspx.page	Begin LoadComplete	0.000407384178715451	0.000016
aspx.page	End LoadComplete	0.000421492117014872	0.000014
aspx.page	Begin PreRender	0.000435181007642033	0.000014
aspx.page	End PreRender	0.000450127041285973	0.000015
aspx.page	Begin PreRenderComplete	0.000464374662142814	0.000014
aspx.page	End PreRenderComplete	0.000478203235327395	0.000014
aspx.page	Begin SaveState	0.000648127066428834	0.000170
aspx.page	End SaveState	0.00611760712604535	0.005469
aspx.page	Begin SaveStateComplete	0.00615560078166359	0.000038
aspx.page	End SaveStateComplete	0.0061717341170456	0.000016
aspx.page	Begin Render	0.00618584205534502	0.000014
aspx.page	End Render	0.0069005977016632	0.000715

Control Tree

Control	Type	Render Size Bytes (including	ViewState Size Bytes (excluding	ControlState Size Bytes (excluding

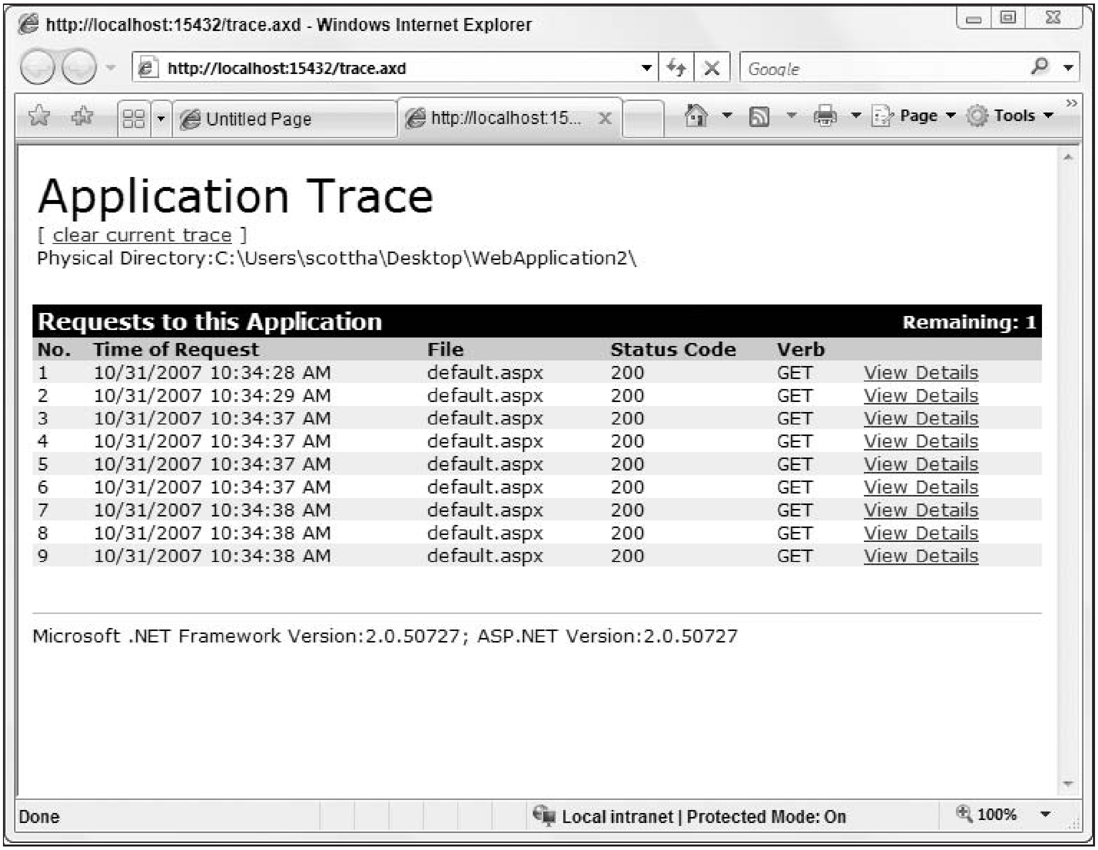
Figure 24-6

Page output of tracing shows only the data collected for the current page request. However, when visiting `http://localhost/yoursite/trace.axd` you'll see detailed data collected for all requests to the site thus far. If you're using the built-in ASP.NET Development Server, remove the current page from the URL and replace it with `trace.axd`. Don't change the automatically selected port or path.

Again, `trace.axd` is an internal handler, not a real page. When it's requested from a local browser, as shown in Figure 24-7, it displays all tracing information for all requests up to a preset limit.

Figure 24-7 shows that nine requests have been made to this application and the right side of the header indicates ''Remaining: 1''. That means that there is one more request remaining before tracing stops for this application. After that final request, tracing data is not saved until an application recycle or until you click ''Clear current trace'' from the `trace.axd` page. The request limit can be raised in `web.config` at the expense of memory:

```
<trace requestLimit="100" pageOutput="true" enabled="true"/>
```


http://localhost:15432/trace.axd - Windows Internet Explorer
http://localhost:15432/trace.axd
Google
Untitled Page
http://localhost:15...
Page
Tools
Done
Local intranet | Protected Mode: On
100%

Application Trace

[clear current trace]

Physical Directory:C:\Users\scottha\Desktop\WebApplication2\

Requests to this Application — Remaining: 1

No.	Time of Request	File	Status Code	Verb	
1	10/31/2007 10:34:28 AM	default.aspx	200	GET	View Details
2	10/31/2007 10:34:29 AM	default.aspx	200	GET	View Details
3	10/31/2007 10:34:37 AM	default.aspx	200	GET	View Details
4	10/31/2007 10:34:37 AM	default.aspx	200	GET	View Details
5	10/31/2007 10:34:37 AM	default.aspx	200	GET	View Details
6	10/31/2007 10:34:37 AM	default.aspx	200	GET	View Details
7	10/31/2007 10:34:38 AM	default.aspx	200	GET	View Details
8	10/31/2007 10:34:38 AM	default.aspx	200	GET	View Details
9	10/31/2007 10:34:38 AM	default.aspx	200	GET	View Details

Microsoft .NET Framework Version:2.0.50727; ASP.NET Version:2.0.50727

Figure 24-7

The maximum request limit value is 10000. If you try to use any greater value, ASP.NET uses 10000 anyway and gives you no error. However, a new property called `mostRecent` was added to the trace section in ASP.NET 2.0. When set to `true`, it shows the most recent requests that are stored in the trace log up to the request limit — instead of showing tracing in the order it occurs (the default) — without using up a lot of memory. Setting `mostRecent` to true causes memory to be used only for the trace information it stores and automatically throws away tracing information over the `requestLimit`.

Clicking View Details from `Trace.axd` on any of these requests takes you to a request-specific page with the same details shown in Figure 24-6.

Tracing from Components

The tracing facilities of ASP.NET are very powerful and can stand alone. However, previously we mentioned `System.Diagnostics.Trace`, the tracing framework in the Base Class Library that is not Web-specific and that receives consistent and complete tracing information when an ASP.NET application calls a non–Web-aware component. This can be confusing. Which should you use?

`System.Diagnostics.Trace` is the core .NET Framework tracing library. Along with `System.Diagnostics.Debug`, this class provides flexible, non-invasive tracing and debug output for any application. But, as mentioned earlier, there is rich tracing built into the `System.Web` namespace. As a Web developer, you'll find yourself using ASP.NET's tracing facilities. You may need to have ASP.NET-specific tracing forwarded to the base framework's `System.Diagnostics.Trace`, or more likely, you'll want to have your non–Web-aware components output their trace calls to ASP.NET so you can take advantage of `trace.axd` and other ASP.NET specific features.

Additionally, some confusion surrounds `Trace.Write` and `Debug.Write` functions. Look at the source code for `Debug.Write`, and you see something like this:

```
 [Conditional("DEBUG")]
public static void Write(string message)
{
    TraceInternal.Write(message);
}
```

Notice that `Debug.Write` calls a function named `TraceInternal.Write`, which has a conditional attribute indicating that `Debug.Write` is compiled only if the debug preprocessor directive was set. In other words, you can put as many calls to `Debug.Write` as you want in your application without affecting your performance when you compile in Release mode. This enables you to be as verbose as you want to be during the debugging phase of development.

`TraceInternal` cycles through all attached trace listeners, meaning all classes that derive from the `TraceListener` base class and are configured in that application's configuration file. The default `TraceListener` lives in the aptly named `DefaultTraceListener` class and calls the Win32 API `OutputDebugString`. `OutputDebugString` sends your string into the abyss and, if a debugger is listening, it is displayed. If no debugger is listening, `OutputDebugString` does nothing. Everyone knows the debugger listens for output from `OutputDebugString` so this can be a very effective way to listen in on debug versions of your application.

For quick and dirty no-touch debugging, try using DbgView from SysInternals at `www.sysinternals.com/ntw2k/freeware/debugview.shtml`. DbgView requires no installation, works great with all your calls to Debug.Writer, and has lots of cool features such as highlighting and logging to a file.

Now, if you look at the source code for `Trace.Write` (that's `TRACE` not `DEBUG`), you see something like this:

```
 [Conditional("TRACE")]
public static void Write(string message)
```

```
{
     TraceInternal.Write(message);
}
```

The only difference between `Debug.Write` and `Trace.Write` given these two source snippets is the conditional attribute indicating the preprocessor directive `TRACE`. You can conditionally compile your assemblies to include tracing statements, debug statements, both, or neither. Most people keep `TRACE` defined even for release builds and use the configuration file to turn tracing on and off. More than likely, the benefits you gain from making tracing available to your users far outweigh any performance issues that might arise.

Because `Trace.Write` calls the `DefaultTraceListener` just like `Debug.Write`, you can use any debugger to tap into tracing information. So, what's the difference?

When designing your application, think about your deployment model. Are you going to ship debug builds or release builds? Do you want a way for end users or systems engineers to debug your application using log files or the event viewer? Are there things you want only the developer to see?

Typically, you want to use tracing and `Trace.Write` for any formal information that could be useful in debugging your application in a production environment. `Trace.Write` gives you everything that `Debug.Write` does, except it uses the `TRACE` preprocessor directive and is not affected by debug or release builds.

This means you have four possibilities for builds: Debug On, Trace On, Both On, or Neither On. You choose what's right for you. Typically, use Both On for debug builds and Trace On for production builds. You can specify these conditional attributes in the property pages or the command line of the compiler, as well as with the C# `#define` keyword or `#CONST` keyword for Visual Basic.

Trace Forwarding

You often find existing ASP.NET applications that have been highly instrumented and make extensive use of the ASP.NET `TraceContext` class. ASP.NET version 2.0 introduces a new attribute to the `web.config <trace>` element that allows you to route messages emitted by ASP.NET tracing to `System.Diagnostics.Trace`: `writeToDiagnosticsTrace`.

```
<trace writeToDiagnosticsTrace="true" pageOutput="true" enabled="true"/>
```

When you set `writeToDiagnosticsTrace` to `true`, all calls to `System.Web.UI.Page.Trace.Write` (the ASP.NET TraceContext) also go to `System.Diagnostics.Trace.Write`, enabling you to use all the standard TraceListeners and tracing options that are covered later in this chapter. The simple `writeToDiagnoticsTrace` setting connects the ASP.NET tracing functionality with the rest of the base class library. I use this feature when I'm deep in debugging my pages, and it's easily turned off using this configuration switch. I believe that more information is better than less, but you may find the exact page event information too verbose. Try it and form your own opinion.

TraceListeners

Output from `System.Diagnostics.Trace` methods is routable by a TraceListener to a text file, to ASP.NET, to an external monitoring system, even to a database. This powerful facility is a woefully underused tool in many ASP.NET developers' tool belts. In ASP.NET 1.1, some component developers who knew their components were being used within ASP.NET would introduce a direct reference to `System.Web` and call `HttpContext.Current.Trace`. They did this so that their tracing information would appear in the developer-friendly ASP.NET format. All components called within the context of an `HttpRequest` automatically receive access to that request's current context, enabling the components to talk directly to the request and retrieve cookies or collect information about the user.

However, assuming an `HttpContext` will always be available is dangerous for a number of reasons. First, you are making a big assumption when you declare that your component can be used only within the context of an HttpRequest. Notice that this is said within the context of *a request*, not within the context of *an application*. If you access `HttpContext.Current` even from within the `Application_Start`, you will be surprised to find that `HttpContext.Current` is null. Second, marrying your component's functionality to `HttpContext` makes it tricky if not impossible to use your application in any non-Web context, and unit testing becomes particularly difficult.

If you have a component that is being used by a Web page, but it also needs to be unit tested outside of Web context or must be called from any other context, don't call `HttpContext.Current.Trace`. Instead, use the standard `System.Diagnostics.Trace` and redirect output to the ASP.NET tracing facilities using the new WebPageTraceListener described in the next section. Using the standard trace mechanism means your component can be used in any context, Web or otherwise. You'll still be able to view the component's trace output with a TraceListener.

The framework comes with a number of very useful TraceListeners; you can add them programmatically or via a `.config` file. For example, you can programmatically add a TraceListener log to a file, as shown in Listing 24-2. These snippets required the `System.Diagnostics` and `System.IO` namespaces.

Listing 24-2: Configuring TraceListeners

VB

```
Dim myTextListener As New TextWriterTraceListener(File.Create("c:\myListener.log"))
Trace.Listeners.Add(myTextListener)
```

C#

```
TextWriterTraceListener myTextListener = new
  TextWriterTraceListener(File.Create(@"c:\myListener.log"));
Trace.Listeners.Add(myTextListener);
```

You can do the same thing declaratively in `web.config` via an `add` element that passes in the type of TraceListener to use, along with any initializing data it might need. TraceListeners already configured in `machine.config` or a parent `web.config` can also be removed using the `remove` tag, along with their name:

```
<configuration>
 <system.diagnostics>
     <trace autoflush="false" indentsize="4">
         <listeners>
            <add name="myListener"
                type="System.Diagnostics.TextWriterTraceListener"
                initializeData="c:\myListener.log" />
            <remove name="Default" />
         </listeners>
     </trace>
 </system.diagnostics>
</configuration>
```

TraceListeners, such as `TextWriterTraceListener`, that access a resource (such as a file, event log, or database) require that the ASP.NET worker process be run as a user who has sufficient access. In order to write to `c:\foo\example.log`, for example, the ASP.NET worker process requires explicit write access in the Access Control List (ACL) of that file.

Notice the preceding example also optionally removes the default TraceListener. If you write your own TraceListener, you must provide a fully qualified assembly name in the `type` attribute.

The New ASP.NET WebPageTraceListener

The new ASP.NET `WebPageTraceListener` derives from `System.Diagnostics.TraceListener` and automatically forwards tracing information from any component calls to `System.Diagnostics.Trace.Write`. This enables you to write your components using the most generic trace provider and to see its tracing output in the context of your ASP.NET application.

The `WebPageTraceListener` is added to the `web.config` as shown in the following example. Note that we use the fully qualified assembly name for System.Web:

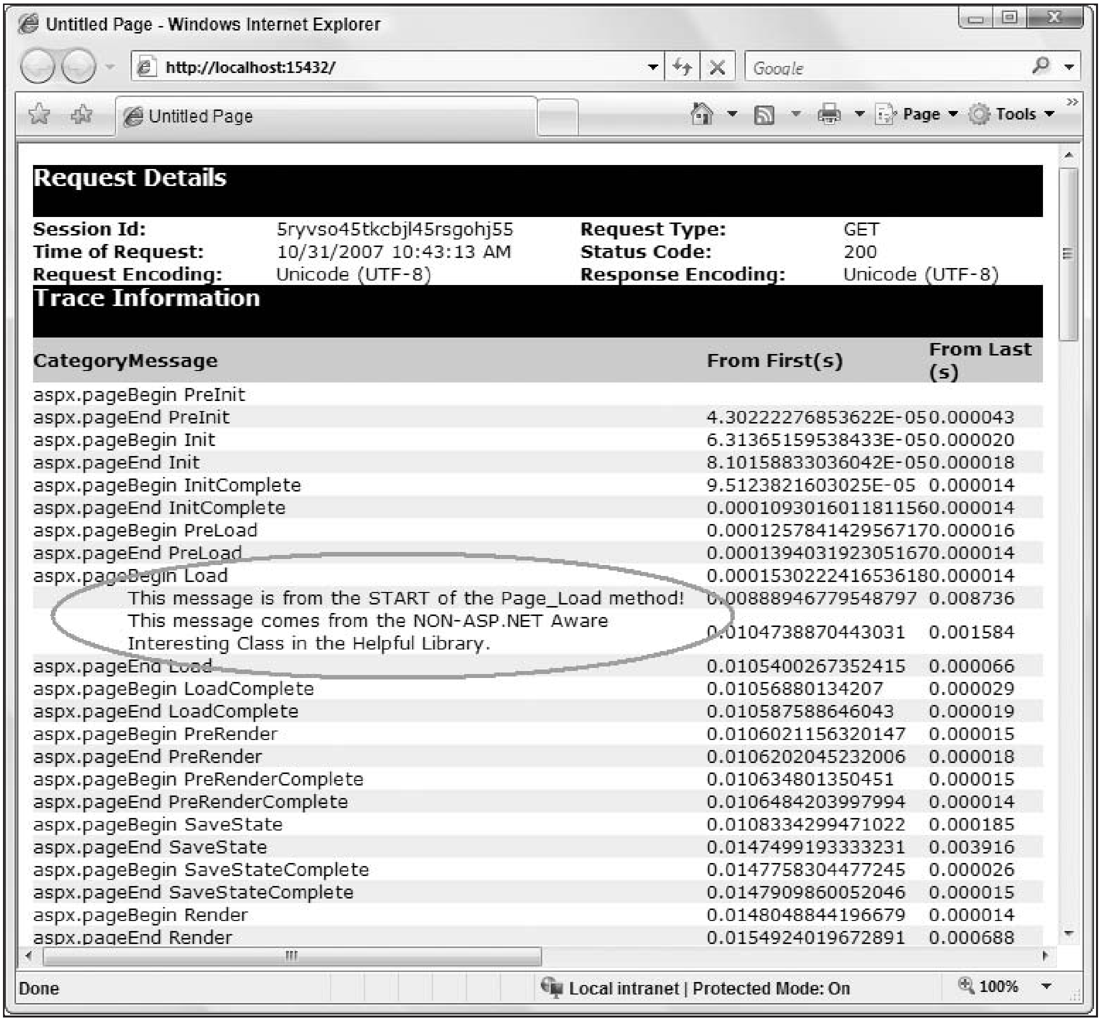
```
<configuration>
 <system.diagnostics>
     <trace autoflush="false" indentsize="4">
         <listeners>
             <add name="webListener"
                type="System.Web.WebPageTraceListener, System.Web, Version=2.0.0.0,
                      Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"/>
         </listeners>
     </trace>
 </system.diagnostics>
 <system.web>
        <trace enabled="true" pageOutput="false" localOnly="true" />
 </system.web>
</configuration>
```

Figure 24-8 shows output from a call to `System.Diagnostics.Trace.Write` from a referenced library. It appears within ASP.NET's page tracing. The line generated from the referenced library is circled in this figure.

EventLogTraceListener

Tracing information can also be sent to the event log using the EventLogTraceListener. This can be a little tricky because ASP.NET requires explicit write access to the event log:

```
<configuration>
 <system.diagnostics>
     <trace autoflush="false" indentsize="4">
         <listeners>
            <add name="EventLogTraceListener"
                type="System.Diagnostics.EventLogTraceListener"
                initializeData="Wrox"/>
         </listeners>
     </trace>
 </system.diagnostics>
</configuration>
```


Untitled Page - Windows Internet Explorer
http://localhost:15432/
Google
Untitled Page
Page
Tools
Done
Local intranet | Protected Mode: On
100%

Request Details

Session Id:	5ryvso45tkcbjl45rsgohj55	**Request Type:**	GET
Time of Request:	10/31/2007 10:43:13 AM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

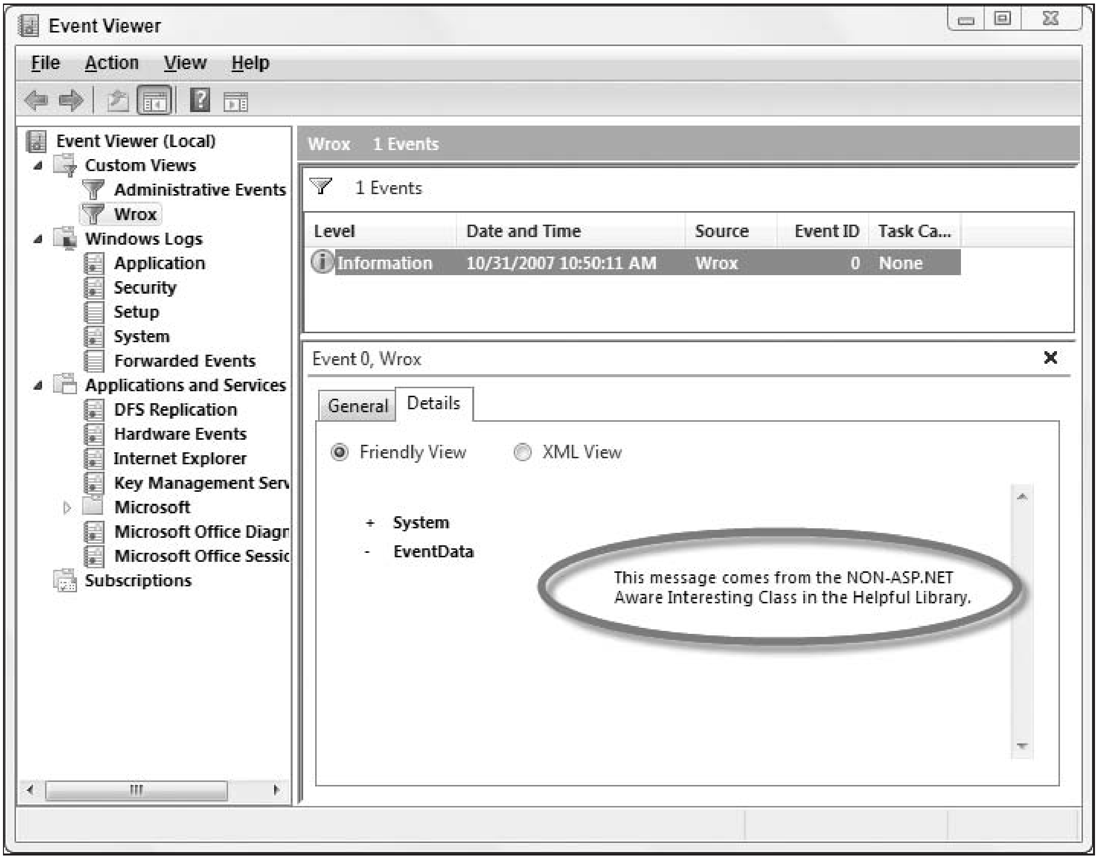
Trace Information

Category	Message	From First(s)	From Last (s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	4.30222276853622E-05	0.000043
aspx.page	Begin Init	6.31365159538433E-05	0.000020
aspx.page	End Init	8.10158833036042E-05	0.000018
aspx.page	Begin InitComplete	9.5123821603025E-05	0.000014
aspx.page	End InitComplete	0.000109301601181156	0.000014
aspx.page	Begin PreLoad	0.000125784142956717	0.000016
aspx.page	End PreLoad	0.000139403192305167	0.000014
aspx.page	Begin Load	0.000153022241653618	0.000014
	This message is from the START of the Page_Load method!	0.00888946779548797	0.008736
	This message comes from the NON-ASP.NET Aware Interesting Class in the Helpful Library.	0.0104738870443031	0.001584
aspx.page	End Load	0.0105400267352415	0.000066
aspx.page	Begin LoadComplete	0.01056880134207	0.000029
aspx.page	End LoadComplete	0.010587588646043	0.000019
aspx.page	Begin PreRender	0.0106021156320147	0.000015
aspx.page	End PreRender	0.0106202045232006	0.000018
aspx.page	Begin PreRenderComplete	0.010634801350451	0.000015
aspx.page	End PreRenderComplete	0.0106484203997994	0.000014
aspx.page	Begin SaveState	0.0108334299471022	0.000185
aspx.page	End SaveState	0.0147499193333231	0.003916
aspx.page	Begin SaveStateComplete	0.0147758304477245	0.000026
aspx.page	End SaveStateComplete	0.0147909860052046	0.000015
aspx.page	Begin Render	0.0148048844196679	0.000014
aspx.page	End Render	0.0154924019672891	0.000688

Figure 24-8

Notice that `"Wrox"` is passed in as a string to the `initializeData` attribute as the TraceListener is added. The string `"Wrox"` appears as the application or *source* for this event. This works fine when debugging your application; most likely, the debugging user has the appropriate access. However, when your application is deployed, it will probably run under a less privileged account, so you must give explicit write access to a registry key such as `HKLM\System\CurrentControlSet\Services\EventLog\Application\Wrox`, where `"Wrox"` is the same string passed in to `initializeData`. Remember that registry keys have ACLs (Access Control Lists) just as files do. Use `RegEdit.exe` to change the permissions on a registry key by right-clicking the key and selecting Properties, and setting the ACL just like you would for a file.

Be careful when using the EventLogTraceListener because your event log can fill up fairly quickly if you have a particularly chatty application. Figure 24-9 shows the same tracing output used in Figure 24-8, this time in the event log. The Event Viewer has changed in Windows Vista and you'll need to create a simple Custom View that shows only ''Wrox'' events.


Event Viewer
File
Action
View
Help
Event Viewer (Local)
Custom Views
Administrative Events
Wrox
Windows Logs
Application
Security
Setup
System
Forwarded Events
Applications and Services
DFS Replication
Hardware Events
Internet Explorer
Key Management Serv
Microsoft
Microsoft Office Diagn
Microsoft Office Sessic
Subscriptions
Wrox 1 Events
1 Events
Event 0, Wrox
General
Details
● Friendly View
○ XML View
+ System
- EventData

Level	Date and Time	Source	Event ID	Task Ca...
Information	10/31/2007 10:50:11 AM	Wrox	0	None

This message comes from the NON-ASP.NET Aware Interesting Class in the Helpful Library.

Figure 24-9

Other Useful Listeners

The .NET 2.0 Framework added two TraceListeners in addition to the WebPageTraceListener:

- **XmlWriterTraceListener:** Derives from `TextWriterTraceListener` and writes out a strongly typed XML file.
- **DelimitedListTraceListener:** Also derives from `TextWriterTraceListener`; writes out comma-separated values (CSV) files.

One of the interesting things to note about the XML created by the XmlWriterTraceListener — it's not well-formed XML! Specifically, it doesn't have a root node; it's just a collection of peer nodes, as shown in the following code. This may seem like it goes against many of the ideas you've been told about XML, but think of each event as a document. Each stands alone and can be consumed alone. They just happen to be next to each other in one file. Certainly, the absence of an ultimate closing tag cleverly dodges the issue of wellformedness and allows easy appending to a file.

```
<E2ETraceEvent xmlns=\"http://schemas.microsoft.com/2004/06/E2ETraceEvent\">
    <System xmlns=\"http://schemas.microsoft.com/2004/06/windows/eventlog/system\">
        <EventID>0</EventID>
        <Type>3</Type>
        <SubType Name="Information">0</SubType>
        <Level>8</Level>
        <TimeCreated SystemTime="2005-11-05T12:43:44.4234234Z">
        <Source Name="WroxChapter21.exe"/>
        <Correlation ActivityID="{00000000-0000-0000-0000-000000000000>
        <Execution ProcessName="WroxChapter21.exe" ProcessID="4234" ThreadID="1"/>
        <Channel/>
        <Computer>SCOTTPC</Computer>
    </System>
    <ApplicationData>Your Text Here</ApplicationData>
</E2ETraceEvent>
<E2ETraceEvent xmlns=\"http://schemas.microsoft.com/2004/06/E2ETraceEvent\">
    <System xmlns=\"http://schemas.microsoft.com/2004/06/windows/eventlog/system\">
        <EventID>0</EventID>
        <Type>3</Type>
...the XML continues...
```

The ‘‘E2E’’ in `E2ETraceEvent` stands for end-to-end. Notice that it includes information such as your computer name and a ‘‘correlation id.’’ Microsoft will include TraceViewer tools with coming products, such as the product codenamed *Indigo* and the managed WinFX API that will consume this XML Schema and help you diagnose problems with operations that span multiple machines in a Web farm. If your ASP.NET application makes calls to an Indigo service, you may want your app to supply its tracing information in this format to make aggregated analysis easier.

The .NET 3.5 Framework added one additional TraceListeners, the IisTraceListener. This new TraceListener has been added to the System.Web Assembly in the .NET 3.5 but that assembly is still versioned as 2.0. Consequently, this class is available only on systems that have the .NET Framework 3.5 installed. Remember that the .NET Framework 3.5 includes fixes for the 2.0 Framework as well as some new functionality, like the `IisTraceListener`.

Much like the `WebPageTraceListener` bridges Diagnostics Tracing with ASP.NET tracing, the `IisTraceListener` bridges the tracing mechanism of ASP.NET with IIS 7.0. This listener lets us raise events to the IIS 7.0 infrastructure. See Chapter 11 on IIS7 for more details on this new class and its use.

Diagnostic Switches

It’s often not convenient to recompile your application just because you want to change tracing characteristics. Sometimes you may want to change your configuration file to add and remove TraceListeners. At other times, you may want to change a configuration parameter or ‘‘flip a switch’’ to adjust the amount of detail the tracing produces. That’s where `Switch` comes in. `Switch` is an abstract base class that supports a series of diagnostic switches that you can control by using the application’s configuration file.

BooleanSwitch

To use a `BooleanSwitch`, create an instance and pass in the switch name that appears in the application’s `config` file (see Listing 24-3).

Listing 24-3: Using diagnostic switches

```
<configuration>
 <system.diagnostics>
  <switches>
   <add name="ImportantSwitch" value="1" /> <!-- This is for the BooleanSwitch -->
   <add name="LevelSwitch" value="3" />     <!-- This is for the TraceSwitch -->
   <add name="SourceSwitch" value="4" />    <!-- This is for the SourceSwitch -->
  </switches>
 </system.diagnostics>
</configuration>
```

Switches can be used in an `if` statement for any purpose, but they are most useful in the context of tracing along with `System.Diagnostics.Trace.WriteIf`:

VB

```
Dim aSwitch As New BooleanSwitch("ImportantSwitch", "Show errors")
System.Diagnostics.Trace.WriteIf(aSwitch.Enabled, "The Switch is enabled!")
```

C#

```
BooleanSwitch aSwitch = new BooleanSwitch("ImportantSwitch", "Show errors");
System.Diagnostics.Trace.WriteIf(aSwitch.Enabled, "The Switch is enabled!");
```

If `ImportantSwitch` is set to `1`, or a non-zero value, in the `config` file, the call to `WriteIf` sends a string to trace output.

TraceSwitch

TraceSwitch offers five levels of tracing from 0 to 4, implying an increasing order: Off, Error, Warning, Info, and Verbose. You construct a `TraceSwitch` exactly as you create a `BooleanSwitch`:

VB

```
Dim tSwitch As New TraceSwitch("LevelSwitch", "Trace Levels")
System.Diagnostics.Trace.WriteIf(tSwitch.TraceInfo, "The Switch is 3 or more!")
```

C#

```
TraceSwitch tSwitch = new TraceSwitch("LevelSwitch", "Trace Levels");
System.Diagnostics.Trace.WriteIf(tSwitch.TraceInfo, "The Switch is 3 or more!");
```

A number of properties on the `TraceSwitch` class return `true` if the switch is at the same level or at a higher level than the property's value. For example, the `TraceInfo` property will return `true` if the switch's value is set to 3 or more.

SourceSwitch

New with .NET 2.0 is `SourceSwitch`, which is similar to `TraceSwitch` but provides a greater level of granularity. You call `SourceSwitch.ShouldTrace` with an EventType as the parameter:

VB

```
Dim sSwitch As New SourceSwitch("SourceSwitch", "Even More Levels")
```

```
System.Diagnostics.Trace.WriteIf(sSwitch.ShouldTrace(TraceEventType.Warning), _
                                "The Switch is 3 or more!")
```

C#

```
SourceSwitch sSwitch = new SourceSwitch("SourceSwitch", " Even More Levels");
System.Diagnostics.Trace.WriteIf(sSwitch.ShouldTrace(TraceEventType.Warning),
                                "The Switch is 4 or more!");
```

Web Events

It doesn't exactly qualify as debugging, but a series of new application-monitoring and health-monitoring tools within the system has been added in ASP.NET's `System.Web.Management` namespace in ASP.NET 2.0. These tools can be as valuable as tracing information in helping you monitor, maintain, and diagnose the health of your application. The system has a whole new event model and event engine that can update your application with runtime details. There are a number of built-in events, including application lifetime events such as start and stop and a heartbeat event. You can take these base classes and events and build on them to create events of your own. For example, you might want to create an event that tells you when a user downloads a particularly large file or when a new user is created in your personalization database. You can have your application send an e-mail to you once a day with statistics.

For instance, you can create your own event by deriving from `System.Web.Management.WebBaseEvent`, as shown in Listing 24-4.

Listing 24-4: Web events

VB

```
Imports System
Imports System.Web.Management

Namespace Wrox
    Public Class WroxEvent
       Inherits WebBaseEvent

        Public Const WroxEventCode As Integer = WebEventCodes.WebExtendedBase + 1
        Public Sub New(ByVal message As String, ByVal eventSource As Object)
            MyBase.New(message, eventSource, WroxEventCode)
        End Sub
    End Class
End Namespace
```

C#

```
namespace Wrox
{
    using System;
    using System.Web.Management;

    public class WroxEvent: WebBaseEvent
    {
        public const int WroxEventCode = WebEventCodes.WebExtendedBase + 1;
        public WroxEvent(string message, object eventSource) :
             base(message, eventSource, WroxEventCode){}
     }
}
```

Later, in a sample `Page_Load`, you raise this event to the management subsystem:

VB

```
Protected Sub Page_Load(sender As Object, e As EventArgs)
    ' Raise a custom event
    Dim anEvent As Wrox.WroxEvent = New Wrox.WroxEvent("Someone visited here", Me)
    anEvent.Raise()
  End Sub
```

C#

```
protected void Page_Load(Object sender, EventArgs e)
{
    // Raise a custom event
    Wrox.WroxEvent anEvent = new Wrox.WroxEvent("Someone visited here!", this);
    anEvent.Raise();
}
```

The event is caught by the management subsystem and can be dispatched to different providers based on a number of rules. This is a much more formal kind of tracing than a call to `Trace.WriteLine`, so you create a strongly typed event class for events specific to your application:

Web.config

```
<?xml version="1.0"?>
<configuration>
    <system.web>
        <healthMonitoring enabled="true">
            <providers>
                <add name="WroxDatabaseLoggingProvider"
                    type="System.Web.Management.SqlWebEventProvider"
                    connectionStringName="QuickStartSqlServer"
                    maxEventDetailsLength="1073741823"
                    buffer="false"/>
            </providers>
            <rules>
                <add
                    name="Application Lifetime Events Rule"
                    eventName="All Events"
                    provider="WroxDatabaseLoggingProvider"
                    profile="Critical" />
            </rules>
        </healthMonitoring>
    </system.web>
</configuration>
```

Debugging

Visual Studio includes two configurations by default: *debug* and *release*. The debug configuration automatically defines the debug and trace constants, enabling your application to provide context to a troubleshooter. The option to generate debugging information is turned on by default, causing a program database (or debug) file (PDB) to be generated for each assembly and your solution. They appear in the same `\bin` folder as your assemblies. Remember, however, that the actual compilation to native code does not occur in Visual Studio, but rather at runtime using just-in-time compilation (JIT). The JIT will automatically optimize your code for speed. Optimized code, however, is considerably harder to debug because the operations that are generated may not correspond directly to lines in your source code. For debug purposes, this option is set to `false`.

What's Required

The PDBs are created when either the C# compiler (`CSC.EXE`) or Visual Basic compiler (`VBC.EXE`) is invoked with the `/debug:full` command lines switch. As an option, if you use `/debug:pdbonly`, you will generate PDBs but still direct the compiler to produce release-mode code.

Debug versus Release

The debug and release configurations that come with Visual Studio are generally sufficient for your needs. However, these configurations control only the compilation options of the code behind files. Remember that, depending on how you've chosen to design your ASP.NET application, the ASP.NET `.aspx` files may be compiled the first time they're hit, or the entire application may compile the first time a page is hit. You can control these compilation settings via the compilation elements within the `<system.web>` section of your application's `web.config`. Set `<compilation debug="true">` to produce binaries as you do when using the `/debug:full` switches. PDBs are also produced.

The average developer is most concerned with the existence of PDB files. When these files exist in your ASP.NET applications `\bin` folder, the runtime provides you with line numbers. Of course, line numbers greatly assist in debugging. You can't step through source code during an interactive debugging session without these files.

An interesting CLR Internals trick: Call `System.Diagnostics.Debugger.Launch` within your assembly, even if the assembly was compiled via `/debug:pdbonly`, and the debugger pops up. The JIT compiler compiles code on the first call to a method, and the code that it generates is debuggable because JIT knows that a debugger is attached.

Debugging and the JIT Dialog

When an unhandled error occurs in an ASP.NET application, the default error handler for the ASP.NET worker process catches it and tries to output some HTML that expresses what happened. However, when you are debugging components outside of ASP.NET, perhaps within the context of unit testing, the debug dialog box appears when the .NET application throws an unhandled exception.

If something has gone horribly wrong with an ASP.NET application, it's conceivable that you may find a Web server with the dialog box popped up waiting for your input. This can be especially inconvenient if the machine has no keyboard or monitor hooked up. The day may come when you want to turn off the debug dialog box that appears, and you have two options to do this:

- You can disable JIT Debugging from the registry. The proper registry key is `HKLM\Software\Microsoft\.NETFramework\DbgJITDebugLaunchSetting`. There are three possible values for the option:
    - `0`: Prompts the user by means of a message box. The choices presented include Continue, which results in a stack dump and process termination, and Attach a Debugger, which means the runtime spawns the debugger listed in the DbgManagedDebugger registry key. If no key exists, the debugger releases control and the process is terminated.

- ❑ `1`: Does not display a dialog box. This results in a stack dump and then process termination.
- ❑ `2`: Launches the debugger listed in the DbgManagedDebugger registry key.

For this option, the registry entry must to be set to `0` *for the dialog box to show up.*

- ❑ To disable the JIT debug dialog box and still present an error dialog box, within Visual Studio.NET, choose Tools ➪ Options ➪ Debugging ➪ Just-In-Time and deselect Common Language Runtime. Instead of the Select a Debugger dialog box, an OK/Cancel dialog box will appear during an unhandled exception.

IIS versus ASP.NET Development Server

ASP.NET greatly simplifies your Web developing experience by enabling you to develop applications without IIS (Internet Information Server — the Web server) on your developer machine. Rather than the traditional style of creating a virtual directory and mapping it to a physical directory, a directory can be opened as a Web site simply by telling Visual Studio that it is a Web site. When you open a Web site from the File menu, the first option on the list of places to open from is the file system. Visual Studio considers any folder that you open to be the root of a Web site. Other options, of course, are opening Web sites from your local IIS instance, FTP, or source control.

Using the IIS option works much as it does in previous versions of Visual Studio with a few convenient changes such as the capability to create a Web site or map a virtual directory directly from the Open Web Site dialog. However, more interesting stuff happens after you open a Web site from the file system.

By default, Web sites that exist only on the file system have a ‘‘just-in-time’’ Web server instantiated called the ASP.NET Development Server. The small Web server hosts the exact same ASP.NET page rendering at runtime that is hosted within IIS on a deployed production site. The page rendering behavior should be identical under the small server as it is under IIS. You should be aware of a few important differences and specific caveats to ensure a smooth transition from development to production.

Create a new Web site by selecting File ➪ New Web Site and immediately pressing F5 to begin a debugging session. You are greeted with a Debugging Not Enabled dialog box. The first option automatically adds a new `web.config` file with debugging enabled. (Earlier versions of Visual Studio required a tedious manual process.) Click OK and balloon help appears in the system tray announcing that the ASP.NET Development Server has started up. It also shows what random high-number port the Web server has selected on the local host. When you close your browser and stop your debugging session, the tiny Web server shuts down.

The ASP.NET Development Server is an application, not a service. It is not a replacement for IIS, nor does it try to be. It’s really just a broker that sits between the developer and the ASP.NET page renderer, and it contains very few, if any, of the security benefits that IIS includes. It is loosely based on a .NET 1.*x* project, code-named Cassini, which is downloadable from `http://asp.net/Projects/Cassini/Download/Default.aspx`. This project was a sample meant to illustrate how to use the `System.Web.Hosting` namespace. The Cassini project was the grandparent, and now the ASP.NET Development Server is a first-class member of the Visual Studio product family. Including this tiny Web server with the Development Environment also allows Visual Studio to be used on Windows XP Home systems that are unable to run IIS.

The small Web server runs under the same user context that runs Visual Studio. If your application requires a specific security context, such as an anonymous user or specific domain user, consider using IIS as your development Web server. Additionally, because the Web server starts up on a port other than port 80, be sure to use best practices while developing your site's navigation scheme. Often, developers assume their site's URL will not include a port number (it will default to port 80), that their site may appear within a specific subdomain (`bar.foo.com`), or that their site will appear within a subdirectory (`www.foo.com/bar`). Consider making your navigation relative to the virtual root of your application so your application is resilient enough to be run in many contexts.

Starting a Debugging Session

There are a number of ways to enter an interactive debugging session with ASP.NET. Visual Studio can fire up the ASP.NET Worker Process, load your newly compiled Web site and attach the debugging to the Worker Process automatically. Or, you can attach a debugger to a site that is already running. Visual Studio also includes a new simpler remote debugging tool for cross-machine debugging.

F5 Debugging

When you start debugging an ASP.NET application, Visual Studio takes into consideration all the Start options within your project properties. Just as ASP.NET 1.*x* Visual Studio can be set to launch the browser on a specific page, the new version allows you to start debugging using the currently selected page. The specific page has been selected so that the Visual Studio debugger can automatically attach the correct process, which might be the Visual Studio Web Server, the ASP.NET Worker Process, or a remote debug monitor.

Attaching to a Process

It's often convenient to jump into an interactive debugging session of a Web site that is already running, and at known state, rather than starting an application from scratch each time you debug. To begin debugging a site that is already running, from Visual Studio's Debug menu, select Attach to Process. The dialog has been improved from previous versions of Visual Studio and now includes a Refresh button and simplifies most common debugging use cases by showing only those processes that belong to the user and that are in the currently running session.

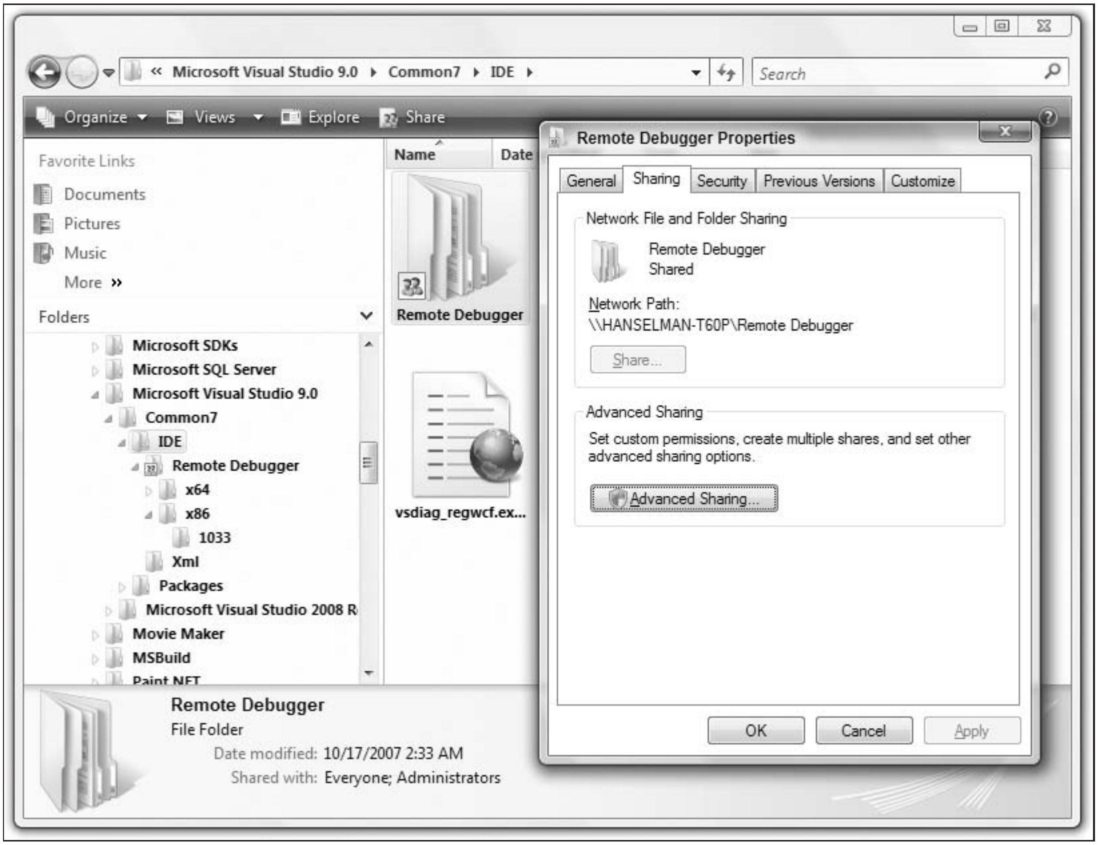
Also included is a transport drop-down with the default transport selected. The default allows you to select processes on your computer or on a remote computer that's running the Remote Debugging Monitor. Other options are there for smart client or unmanaged debugging.

> *The only difference between starting a Debug session via F5 and attaching to a process manually is that when you debug via F5, Visual Studio automatically starts up a browser or external application for you. Remember that if you use Attach to Process, it is assumed that you have already done the work of starting up the process. The ASP.NET Worker Processes under IIS will start up when the site has been hit with an* `HttpRequest` *at least once. The debugger can now attach to the running Worker Process.*

Sometimes you want to debug an ASP.NET application that is already running on your computer. If that application was not started by Visual Studio and you want to attach to it, select Attach to Process from the Debug menu and choose either `ASPNET_WP.exe` (if you're running Windows XP) or `W3WP.exe` (if you are running Windows 2003 server). Be careful that you are not trying to debug an application that is actively servicing other users or you may ruin their experience.

Simpler Remote Debugging

Remote debugging got simpler with Visual Studio 2005. However, in the interest of security, you must have the appropriate credentials to perform remote debugging. You'll find a `Remote Debugger` folder in `C:\Program Files\Microsoft Visual Studio 8\Common7\IDE`. In Figure 24-10, Explorer is shown open and the `Remote Debugger` folder is selected and has been configured as a shared directory for access over the local network.


« Microsoft Visual Studio 9.0 ▸ Common7 ▸ IDE ▸
Search
Organize
Views
Explore
Share
Favorite Links
Documents
Pictures
Music
More »
Folders
Microsoft SDKs
Microsoft SQL Server
Microsoft Visual Studio 9.0
Common7
IDE
Remote Debugger
x64
x86
1033
Xml
Packages
Microsoft Visual Studio 2008 R
Movie Maker
MSBuild
Paint.NET
Name
Date
Remote Debugger
vsdiag_regwcf.ex...
Remote Debugger Properties
General
Sharing
Security
Previous Versions
Customize
Network File and Folder Sharing
Share...
Advanced Sharing
Advanced Sharing...
OK
Cancel
Apply

Remote Debugger
Shared

Network Path:
\\HANSELMAN-T60P\Remote Debugger

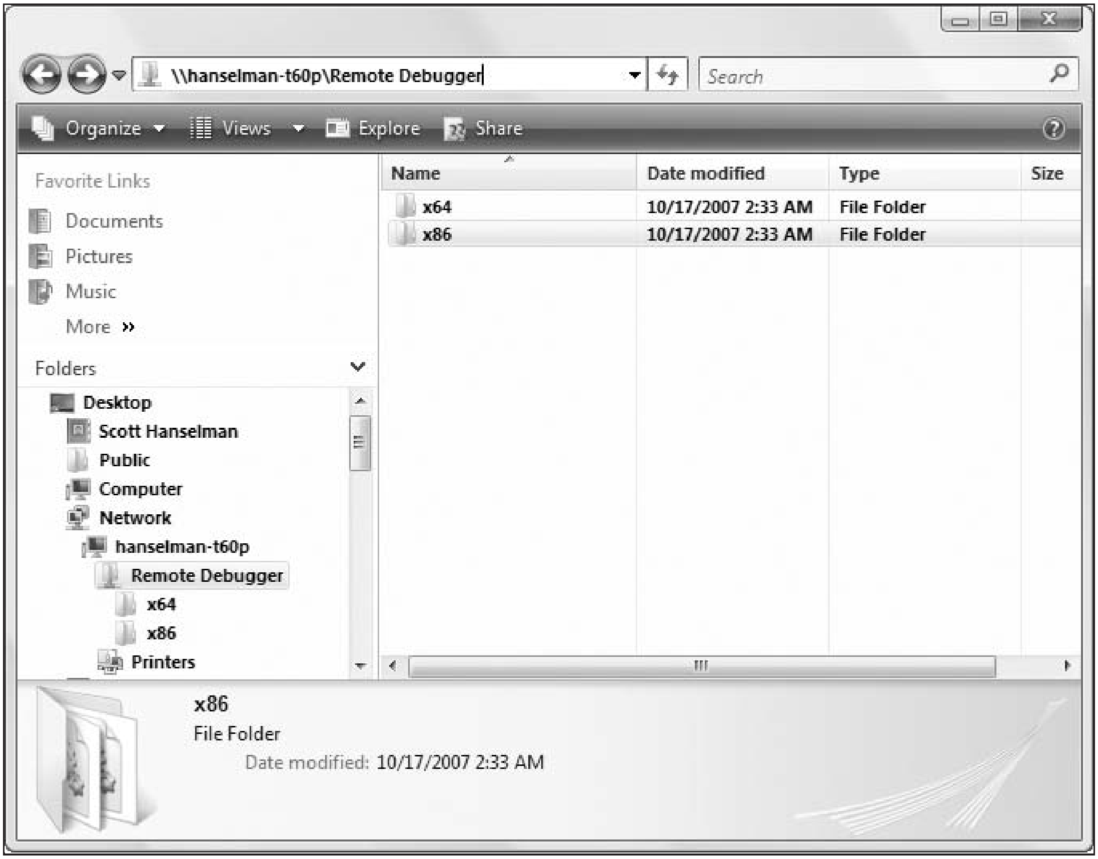
Set custom permissions, create multiple shares, and set other advanced sharing options.

Remote Debugger
File Folder
Date modified: 10/17/2007 2:33 AM
Shared with: Everyone; Administrators

Figure 24-10

To begin, remote debugging must be set up on the machine that contains the application you want to debug. Rather than performing a complicated installation, you can now use the Remote Debug Monitor, and an application that can simply be run off a file share. The easiest scenario has you sharing these components directly from your Visual Studio machine and then running `msvsmon.exe` off the share, as shown in Figure 24-11.

Simply running the Remote Debug Monitor executable off the file share can make remote ASP.NET debugging of an already-deployed applications much simpler, although you still need to manually attach to the ASP.NET worker process because automatic attaching is not supported. Do note that there are now two versions of the debugger, one for x86 processes and one for x64 processes, so make sure you're using the right debugger for your process.



\\hanselman-t60p\Remote Debugger

Name	Date modified	Type	Size
x64	10/17/2007 2:33 AM	File Folder	
x86	10/17/2007 2:33 AM	File Folder	

x86
File Folder
Date modified: 10/17/2007 2:33 AM

Search
Organize
Views
Explore
Share
Favorite Links
Documents
Pictures
Music
More »
Folders
Desktop
Scott Hanselman
Public
Computer
Network
hanselman-t60p
Remote Debugger
x64
x86
Printers

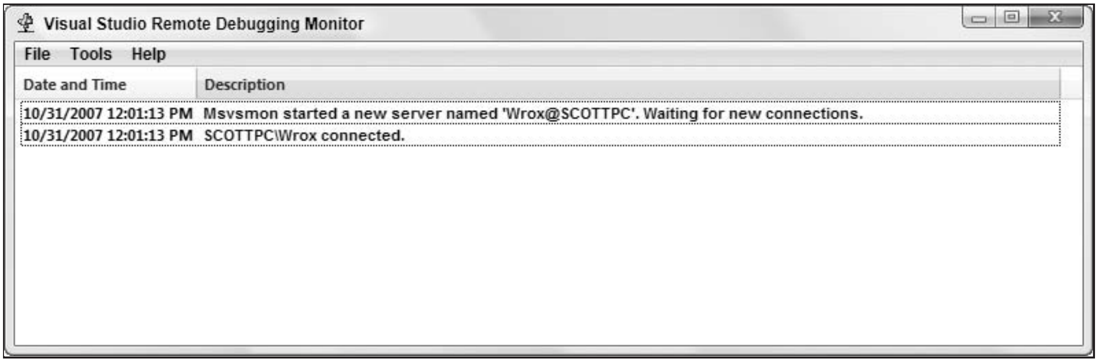
Figure 24-11

You are allowed to debug a process that's running under your account and password without any special permissions. If you need to debug a process running under another account name, such as an ASP.NET worker process running as a user who is not you, you must be an administrator on the machine running the process.

The most important thing to remember when debugging remotely is this: You need to get the user account that is running as Visual Studio to map somehow to a legitimate user account on the machine running the Remote Debug Monitor (`msvsmon.exe`) machine and vice versa. The easiest way to do this is to create a local user account on both computers with the same username and password. To run `msvsmon` as a user other than Visual Studio, you must create two user accounts on each computer.

If one of the machines is located on a domain, be aware that domain account can be mapped to a local account. You create a local user account on both computers. However, if you pick the same username and password as your domain account, Visual Studio can be run as a domain account. Figure 24-12 shows the machine name is SCOTTPC and the username is Wrox. A Wrox user was created on both machines with the same password on each.

For Windows XP machines on a workgroup, the security option entitled Network Security: Shared and Security Model for Local Accounts affects your use of the Remote Debug Monitor. If this option is set to Guest Only — Local Users Authenticate As Guest, then remote debugging fails and shows you a dialog box. Configure this via the Local Security Policy MMC-based administrative tool. The warning doesn't affect Windows 2000 or Windows Server 2003, Windows XP, or Vista-based computers that are joined to a domain.



```
Date and Time           Description
10/31/2007 12:01:13 PM  Msvsmon started a new server named 'Wrox@SCOTTPC'. Waiting for new connections.
10/31/2007 12:01:13 PM  SCOTTPC\Wrox connected.
```

Figure 24-12

Debugging Running Windows XP Service Pack 2

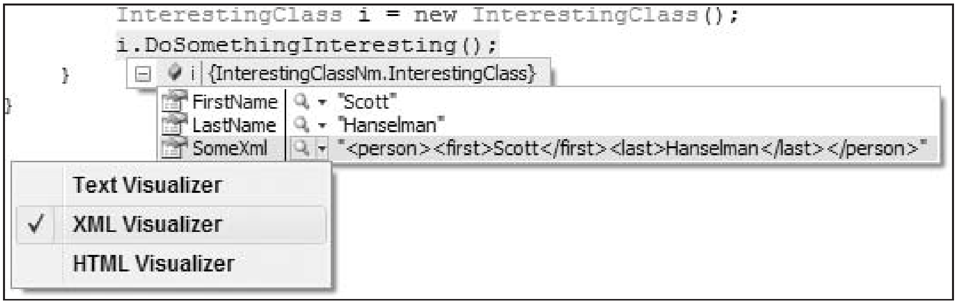
Make sure that TCP Port 80 is set to allow ASP.NET and IIS to communicate with the remote machine. Try to keep the scope limited, using options such as Local Subnet Only, and avoid exposing your Web server to the Internet during development. Also include TCP Port 135, which allows DCOM to communicate with remote machines as well as with UDP Ports 4500 and 500 for IPSec-based security. Last, confirm that the Remote Debug Monitor (`msvsmon.exe`) is in the list of exceptions to the firewall. Again, avoid exposing your debugging session to the outside world. Remote debugging is usually a last resort if the bug isn't reproducible for whatever reason on the developer workstation.

New Tools to Help You with Debugging

The debugging experience in Visual Studio has improved arguably more than any other aspect of the environment. A number of new tools, some obvious, some more subtle, assist you in every step of the debug session.

Debugger Datatips

Previous versions of Visual Studio gave you tooltips when the user hovered the mouse over variables of simple types. Visual Studio 2005 offers *datatips*, allowing. complex types to be explored using a modeless tree-style view that acts like a tooltip and provides much more information. After you traverse the tree to the node that you're interested in, that simple type can be viewed using a visualizer by clicking the small magnifying glass icon, as seen in Figure 24-13.



```
        InterestingClass i = new InterestingClass();
        i.DoSomethingInteresting();
    }
}
```

i {InterestingClassNm.InterestingClass}

FirstName "Scott"

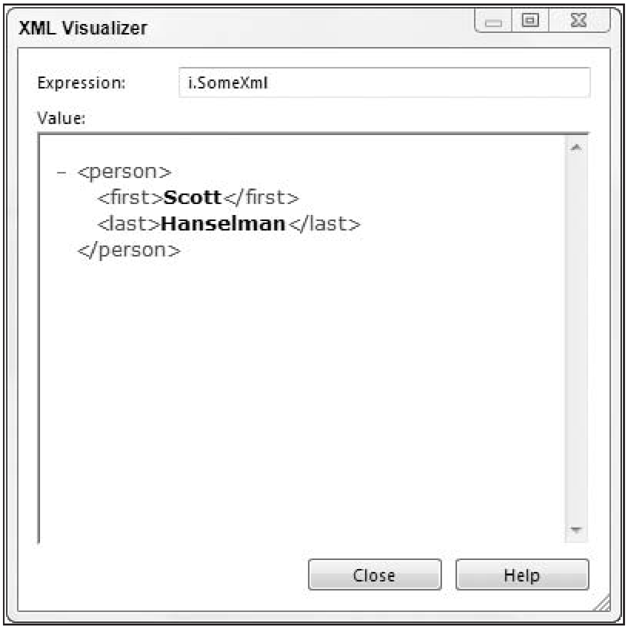
LastName "Hanselman"

SomeXml "<person><first>Scott</first><last>Hanselman</last></person>"

Figure 24-13

Data Visualizers

As you see in Figure 24-13, a simple type can be viewed using any number of data visualizers. For example, if a simple variable such as a string contains a fragment of XML, you might want to visualize that data in a style that's more appropriate for the data's native format, as shown in Figure 24-14.



Expression: i.SomeXml

Value:

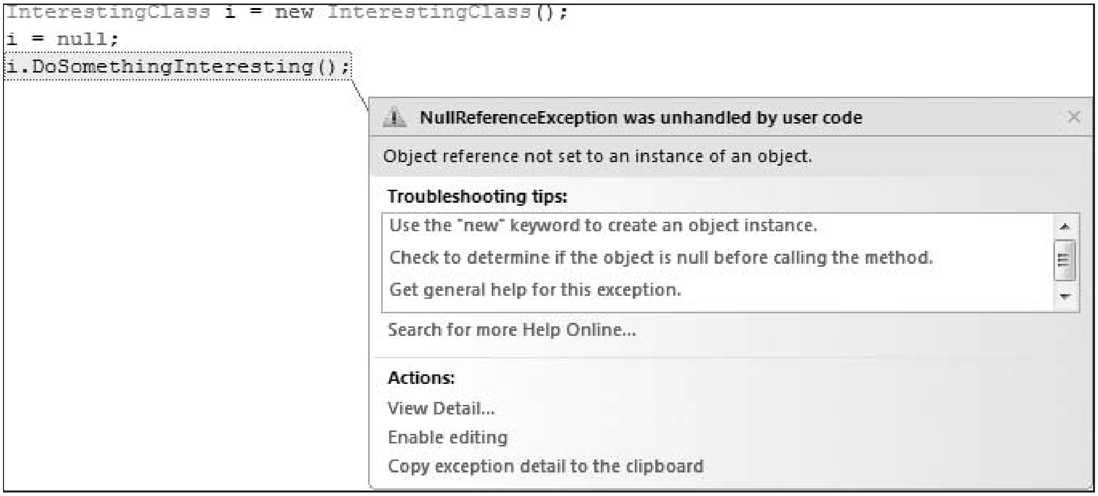
```
- <person>
    <first>Scott</first>
    <last>Hanselman</last>
  </person>
```

Figure 24-14

The visualizers are straightforward to write and, although Visual Studio ships with default visualizers for text, HTML, XML, and DataSets, expect to see a flood of new visualizers appearing on the Internet with support for images, collection classes, and more. The result is a rich, unparalleled debugging experience.

Error Notifications

During an interactive debugging session, Visual Studio now strives to assist you with informative Error Notifications. These notifications not only report on events such as unhandled exceptions, but also offer context-sensitive troubleshooting tips and next steps for dealing with the situation. Figure 24-15 shows an unhandled `NullReferenceException` along with the good advice that we might try using the ''new'' keyword to create an object instance before using it. Oops!



```
InterestingClass i = new InterestingClass();
i = null;
i.DoSomethingInteresting();
```

NullReferenceException was unhandled by user code

Object reference not set to an instance of an object.

Troubleshooting tips:

Use the "new" keyword to create an object instance.

Check to determine if the object is null before calling the method.

Get general help for this exception.

Search for more Help Online...
Actions:
View Detail...
Enable editing
Copy exception detail to the clipboard

Figure 24-15

Edit and Continue (Lack of) Support, or Edit and Refresh

Visual Basic 6 was all about developing things quickly, and its most powerful feature was the Edit and Continue feature, which gave you capability to change code during a debugging session without restarting the session. In break mode, you could modify code fix bugs and move on. The 2.0 version of the CLR has restored this feature for both C# and Visual Basic. Although this has a large number of developers cheering, unfortunately this feature is not available to ASP.NET developers.

In ASP.NET, your assembly is compiled not by Visual Studio, but by the ASP.NET runtime using the same technique it does during a normal Web page request by a browser. To cooperate with the debugger and support Edit and Continue within ASP.NET, a number of fantastically complex modifications to ASP.NET runtime would have been required by the development team. Rather than including support for this feature, ASP.NET developers can use *page recycling*.

This means that code changes are made during a debugging session, and then the whole page is refreshed via F5, automatically recompiled, and re-executed. Basically, ASP.NET 2.0 includes much improved support for Edit and Refresh, but not for Edit and Continue.

Just My Code Debugging

A new concept in the .NET 2.0 CLR is called *Just My Code* debugging. Any method in code can be explicitly marked with the new attribute `[DebuggerNonUserCode]`. Using this explicit technique and a number of other heuristic methods internal to the CLR, the debugger silently skips over code that isn't important to the code at hand. You can find the new preference Enable Just My Code in Tools ➪ Options ➪ Debugging.

The `[DebuggerHidden]` attribute is still available in .NET 2.0 and hides methods from the debugger, regardless of the user's Just My Code preference. The 1.1 attribute `[DebuggerStepThrough]` tells the debugger to step through, rather than into, any method to which it's applied; the `[DebuggerNonUserCode]` attribute is a much more pervasive and complete implementation that works at runtime on delegates, virtual functions, and any arbitrarily complex code.

Be aware that these attributes and this new user option exist to help you debug code effectively and not be fooled by any confusing call stacks. While these can be very useful, be sure not to use them on your components until you're sure you won't accidentally hide the very error you're trying to debug. Typically these attributes are used for components such as proxies or thin shim layers.

Tracepoints

Breakpoints by themselves are useful for stopping execution either conditionally or unconditionally. Standard breakpoints break always. Conditional breakpoints cause you to enter an interactive debugging session based on a condition. Tracing is useful to output the value of a variable or assertion to the debugger or to another location. If you combine all these features, what do you get? Tracepoints, a new and powerful Visual Studio feature. Tracepoints can save you from hitting breakpoints dozens of times just to catch an edge case variable value. They can save you from covering your code with breakpoints to catch a strange case.

To insert a Tracepoint, right-click in the code editor and select Breakpoint ➪ Insert Tracepoint. You'll get the dialog shown in Figure 24-16. The icon that indicates a breakpoint is a red circle, and the icon for a Tracepoint is a red diamond. Arbitrary strings can be created from the dialog using pseudo-variables in the form of keywords such as `$CALLSTACK` or `$FUNCTION`, as well as the values of variables in scope placed in curly braces. In Figure 24-16, the value of `i.FirstName` (placed in curly braces) is shown in the complete string with the Debug output of Visual Studio.

Client-side Javascript Debugging

Excellent client-side Javascript Debugging is new in Visual Studio 2008. If you run an ASP.NET application in a debugging session in Internet Explorer you'll need to enable script debugging. If not, you'll receive a dialog similar to the one in Figure 24-17.

After you've turned on Script Debugging, try a simple ASPX page with some Javascript that changes the text in a textbox to UPPERCASE when the button is pressed.

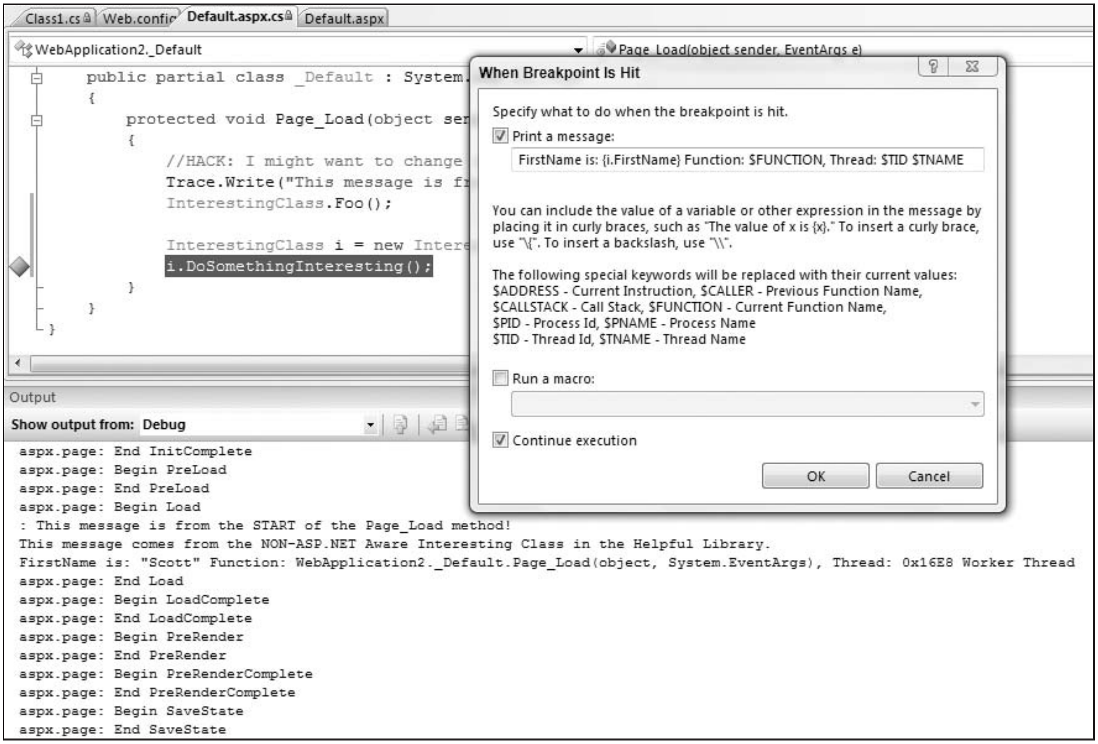
Listing 24-5: Simple Javascript debugging test

ASPX

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <script type="text/javascript">
       function MakeItUpper()
```

Continued

```
            {
                newText = document.getElementById("TextBox1").value.toUpperCase();
                document.getElementById("TextBox1").value = newText;
             }
        </script>
    </head>
    <body>
        <form id="form1" runat="server">
            <div>
                <input type="button" id="Button1" value="Upper"
                    onclick="javascript:MakeItUpper()" />
                <input type="text" id="TextBox1" runat="server"/>
            </div>
        </form>
    </body>
 </html>
```



```
public partial class _Default : System.
{
    protected void Page_Load(object ser
    {
        //HACK: I might want to change
        Trace.Write("This message is fr
        InterestingClass.Foo();

        InterestingClass i = new Intere
        i.DoSomethingInteresting();
    }
}
}
```

When Breakpoint Is Hit

Specify what to do when the breakpoint is hit.

☑ Print a message:

FirstName is: {i.FirstName} Function: $FUNCTION, Thread: $TID $TNAME

You can include the value of a variable or other expression in the message by placing it in curly braces, such as "The value of x is {x}." To insert a curly brace, use "\{". To insert a backslash, use "\\".

The following special keywords will be replaced with their current values:
$ADDRESS - Current Instruction, $CALLER - Previous Function Name,
$CALLSTACK - Call Stack, $FUNCTION - Current Function Name,
$PID - Process Id, $PNAME - Process Name
$TID - Thread Id, $TNAME - Thread Name

☐ Run a macro:

☑ Continue execution

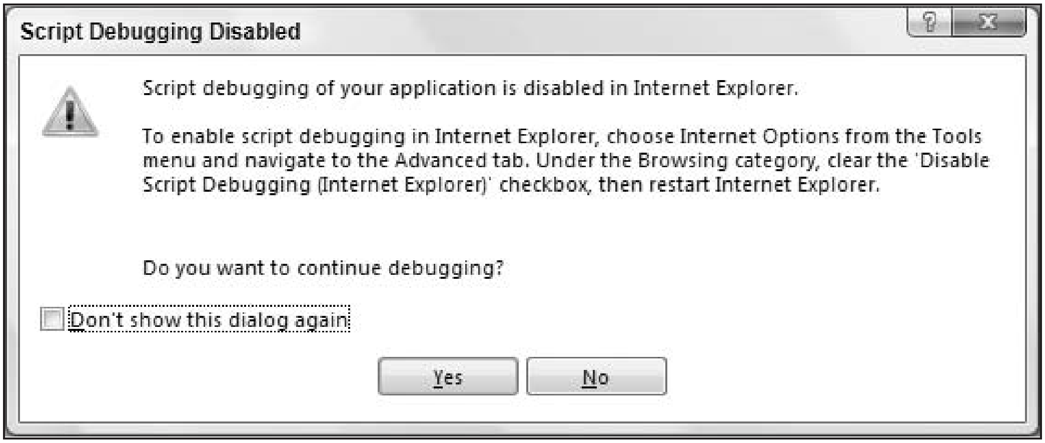
Show output from: Debug

```
aspx.page: End InitComplete
aspx.page: Begin PreLoad
aspx.page: End PreLoad
aspx.page: Begin Load
: This message is from the START of the Page_Load method!
This message comes from the NON-ASP.NET Aware Interesting Class in the Helpful Library.
FirstName is: "Scott" Function: WebApplication2._Default.Page_Load(object, System.EventArgs), Thread: 0x16E8 Worker Thread
aspx.page: End Load
aspx.page: Begin LoadComplete
aspx.page: End LoadComplete
aspx.page: Begin PreRender
aspx.page: End PreRender
aspx.page: Begin PreRenderComplete
aspx.page: End PreRenderComplete
aspx.page: Begin SaveState
aspx.page: End SaveState
```

Figure 24-16

Put a breakpoint on one of the lines of *client-side* Javascript. Note that this is code that runs in the browser, not on the Web server. Start a debugging session with the page from Listing 24-5. Visual Studio will break at that point, as shown in Figure 24-18.

The Javascript debugger in Visual Studio 2008 supports variable tooltips, visualizers, call stacks, locals, watches, and all the features you're used to when debugging .NET-based languages.



Script Debugging Disabled

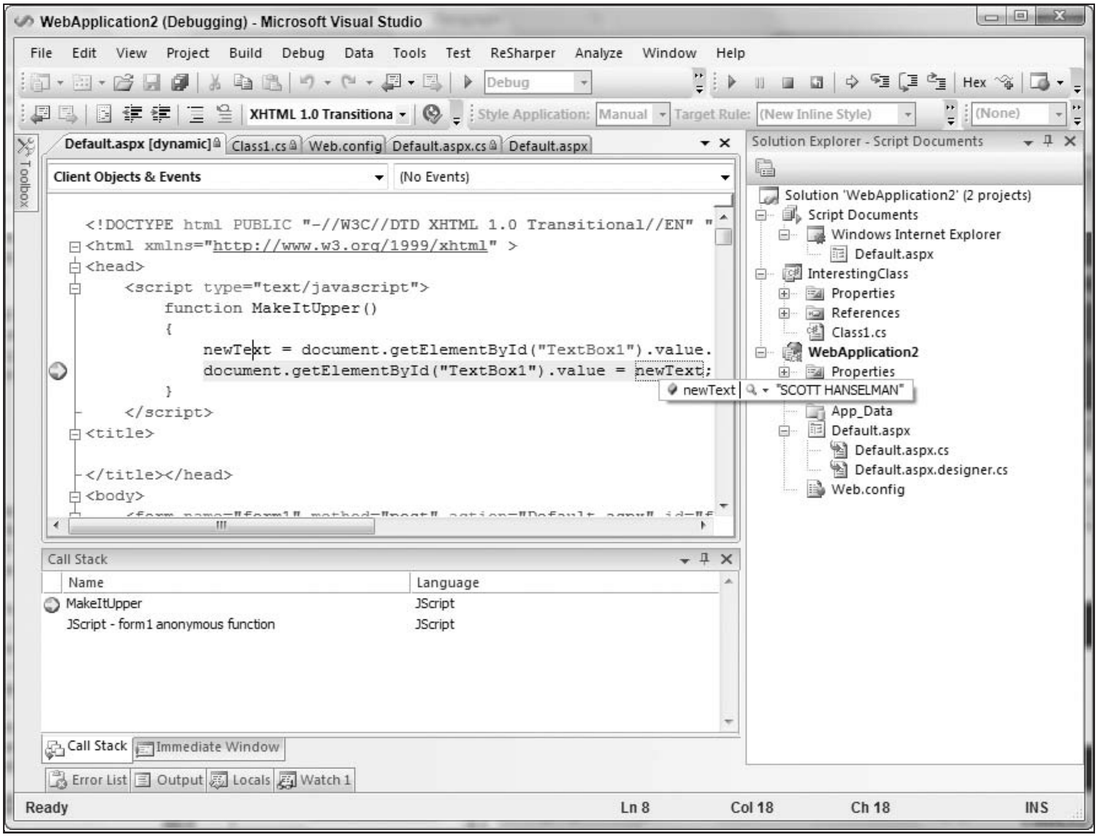
Script debugging of your application is disabled in Internet Explorer.

To enable script debugging in Internet Explorer, choose Internet Options from the Tools menu and navigate to the Advanced tab. Under the Browsing category, clear the 'Disable Script Debugging (Internet Explorer)' checkbox, then restart Internet Explorer.

Do you want to continue debugging?

☐ Don't show this dialog again

Figure 24-17



WebApplication2 (Debugging) - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test ReSharper Analyze Window Help
Debug
Hex
XHTML 1.0 Transitiona
Style Application: Manual
Target Rule: (New Inline Style)
(None)
Toolbox
Default.aspx [dynamic]
Class1.cs
Web.config
Default.aspx.cs
Default.aspx
Client Objects & Events
(No Events)
Solution Explorer - Script Documents
Solution 'WebApplication2' (2 projects)
Script Documents
Windows Internet Explorer
Default.aspx
InterestingClass
Properties
References
Class1.cs
WebApplication2
Properties
App_Data
Default.aspx
Default.aspx.cs
Default.aspx.designer.cs
Web.config
Call Stack
Immediate Window
Error List
Output
Locals
Watch 1

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <script type="text/javascript">
        function MakeItUpper()
        {
            newText = document.getElementById("TextBox1").value.
            document.getElementById("TextBox1").value = newText;
        }
    </script>
<title>

</title></head>
<body>
```

newText "SCOTT HANSELMAN"

Call Stack

Name	Language
MakeItUpper	JScript
JScript - form1 anonymous function	JScript

Ready Ln 8 Col 18 Ch 18 INS

Figure 24-18

Notice in the figure that `Default.aspx` has the word [dynamic] listed on the tab, indicating that this isn't the same `default.aspx` that was edited earlier; you can see that `default.aspx` listed on the final tab. Rather, this is the dynamically generated `default.aspx` that was delivered to the browser, including ViewState and other generated elements. Dynamically generated documents and scripts appear during the debug session in the Solution Explorer.

This rich debugging support on the client side makes creating today's Javascript-heavy AJAX applications much easier.

SQL Stored Proc Debugging

Database projects are file-based projects that let you manage and execute database queries. You can add your existing SQL scripts to the project or create new ones and edit them within Visual Studio. Database projects and SQL debugging are not available in the Express or Standard versions of Visual Studio. They are available only in the Professional or Team Edition Visual Studio SKUs/versions.

When debugging database applications, you can't use Step Into (F11) to step between code in the application tier into the code in SQL Server 2005 (be it T-SQL or CLR SQL). However, you can set a breakpoint in the stored procedure code and use Continue (F5) to execute code to that set break point.

When debugging SQL on SQL Server 2005, be aware of any software or hardware firewalls you may be running. Windows XP SP2's software firewall will warn you what you're trying to do. Be sure to select ''unblock'' in any warning dialogs to ensure that SQL Server 2005 and Visual Studio can communicate.

If you are using a SQL account to connect to the SQL Server, make sure the Windows User Account you run Visual Studio under is also an administrator on the SQL Server machine. You can add accounts to SQL Server's `sysadmin` privilege using the SQL command `sp_addsrvrolemember 'Domain\Name', 'sysadmin'`. Of course, never do this in production; and better yet, do your debugging on a machine with everything installed locally.

If you're using the NT Authentication model on the SQL Server 2005, make sure that account has permissions to run the `sp_enable_sql_debug` stored procedure. You can give account access to this stored procedure by using the SQL commands `CREATE USER UserName FOR LOGIN 'Domain\Name'` followed by `GRANT EXECUTE ON sp_enable_sql_debug TO UserName`. This creates a SQL user that is associated directly with a specific Windows User and then explicitly grants permissions to debug TSQL to that user. On SQL Server 2000, the user must have access to the extended stored procedure `sp_sdidebug`.

For slightly older installations such as Windows 2000 and Windows NT 4, or if you are using SQL 2000, be sure to visit MSDN for the latest details and tools in this space. The MSDN URL for debugging SQL Server is `http://msdn2.microsoft.com/library/zefbf0t6`.

Exception and Error Handling

When an exception occurs in your ASP.NET application code, you can handle it in a number of ways, but the best approach is a multi-pronged one:

- Catch what you expect:
  - Use a `Try/Catch` around error-prone code. This can always catch specific exceptions that you can deal with, such as `System.IO.FileNotFoundException`

    - Rather than catching exceptions around specific chunks of code at the page level, consider using the page-level error handler to catch specific exceptions that might happen anywhere on the page.
- But prepare for unhandled exceptions:
    - Set the `Page.Error` property if a specific page should show a specific error page for any unhandled exception. This can also be done using the `<%@ Page >` directive or the code behind the property.
    - Have default error pages for 400 and 500 errors set in your `web.config`.
    - Have a boilerplate `Application_OnError` handler that takes into consideration both specific exceptions that you can do something about, as well as all unhandled exceptions that you may want logged to either the event log, a text file, or other instrumentation mechanism.

The phrase *unhandled exception* may be alarming, but remember that you don't do anyone any good catching an exception that you can't recover from. Unhandled exceptions are okay if they are just that — exceptional. For these situations, rely on global exception handlers for logging and friendly error pages that you can present to the user.

> **Why try to catch an exception by adding code everywhere if you can catch and log exceptions all in one place? A common mistake is creating a try/catch block around some arbitrary code and catching the least specific exception type — `System.Exception`. A rule of thumb is, don't catch any exception that you can't do anything about. Just because an exception *can* be thrown by a particular method doesn't mean you have to catch it. It's *exceptional*, remember? Also, there are exception handlers at both the page and the application level. Catch exceptions in these two centralized locations rather than all over.**

Handling Exceptions on a Page

To handle exceptions at a page level, override the `OnError` method that `System.Web.UI.Page` inherits from the `TemplateControl` class (see Listing 24-5). Calling `Server.GetLastError` gives you access to the exception that just occurred. Be aware that a chain of exceptions may have occurred, and you can use the `ExceptionGetBaseException` method to return the root exception.

Listing 24-6: Page-level error handling

VB

```
Protected Overrides Sub OnError(ByVal e As System.EventArgs)
    Dim AnError As System.Exception = Server.GetLastError()
    If (TypeOf AnError.GetBaseException() Is SomeSpecificException) Then
        Response.Write("Something bad happened!")
        Response.StatusCode = 200
        Server.ClearError()
        Response.End()
    End If
End Sub
```

Continued

C#

```
protected override void OnError(EventArgs e)
{
    System.Exception anError = Server.GetLastError();
    if (anError.GetBaseException() is SomeSpecificException)
    {
        Response.Write("Something bad happened!");
        Response.StatusCode = 200;
        Server.ClearError();
        Response.End();
    }
}
```

Handling Application Exceptions

The technique of catching exceptions in a centralized location can be applied to error handling at the application level in `Global.asax`, as shown in Listing 24-6. If an exception is not caught on the page, the `web.config` is checked for an alternate error page; if there isn't one, the exception bubbles up to the application and your user sees a complete call stack.

Listing 24-7: Application-level error handling

VB

```
Protected Sub Application_Error(sender as Object, ByVal e As System.EventArgs)
    Dim bigError As System.Exception = Server.GetLastError()
    'Example checking for HttpRequestValidationException
    If (TypeOf bigError.GetBaseException() Is HttpRequestValidationException) Then
        System.Diagnostics.Trace.WriteLine(bigError.ToString)
        Server.ClearError()
    End If
End Sub
```

C#

```
protected void Application_Error(Object sender, EventArgs e)
{
   System.Exception bigError = Server.GetLastError();
   //Example checking for HttpRequestValidationException
   if(bigError.GetBaseException() is HttpRequestValidationException )
   {
     System.Diagnostics.Trace.WriteLine(bigError.ToString());
    Server.ClearError();
   }
}
```

Unhandled application errors turn into HTTP Status Code 500 and display errors in the browser. These errors, including the complete callstack and other technical details, may be useful during development, but are hardly useful at production time. Most often, you want to create an error handler (as shown previously) to log your error and to give the user a friendlier page to view.

If you ever find yourself trying to catch exceptions of type `System.Exception`, take a look at the code to see whether you can avoid it. There's almost never a reason to catch such a non-specific exception, and you're more likely to swallow exceptions that can provide valuable debugging. Check the API documentation for the framework method you are calling — a section specifically lists what exceptions an API call might throw. Never rely on an exception occurring to get a standard code path to work.

Http Status Codes

Every `HttpRequest` results in an `HttpResponse`, and every `HttpResponse` includes a status code. The following table describes 11 particularly interesting HTTP status codes.

Status Code	Explanation
200 OK	Everything went well.
301 Moved Permanently	Reminds the caller to use a new, permanent URL rather than the one he used to get here.
302 Found	Returned during a Response.Redirect. This is the way to say ''No, no, look over here right now.''
304 Not Modified	Returned as the result of a conditional `GET` when a requested document hasn't been modified. It is the basis of all browser-based caching. An HTTP message-body must not be returned when using a 304.
307 Temporary Redirect	Redirects calls to ASMX Web services to alternate URLs. Rarely used with ASP.NET.
400 Bad Request	Request was malformed.
401 Unauthorized	Request requires authentication from the user.
403 Forbidden	Authentication has failed, indicating that the server understood the requests but cannot fulfill it.
404 Not Found	The server has not found an appropriate file or handler to handle this request. The implication is that this may be a temporary state. This happens in ASP.NET not only because a file cannot be found, but also because it may be inappropriately mapped to an IHttpHandler that was not available to service the request.
410 Gone	The equivalent of a permanent 404 indicating to the client that it should delete any references to this link if possible. 404s usually indicate that the server does not know whether the condition is permanent.

Status Code	Explanation
500 Internal Server Error	The official text for this error is ‘‘The server encountered an unexpected condition which prevented it from fulfilling the request,’’ but this error can occur when any unhandled exception bubbles all the way up to the user from ASP.NET.

Any status code greater than or equal to 400 is considered an error and, unless you configure otherwise, the user will likely see an unfriendly message in his browser. If you have not already handled these errors inside of the ASP.NET runtime by checking their exception types, or if the error occurred outside of ASP.NET and you want to show the user a friendly message, you can assign pages to any status code within `web.config`, as the following example shows:

```
<customErrors mode ="On" >
    <error statusCode ="500" redirect ="FriendlyMassiveError.aspx" />
</customErrors>
```

After making a change to the customer errors section of your `web.config`, make sure a page is available to be shown to the user. A classic mistake in error redirection is redirecting the user to a page that will cause an error, thereby getting him stuck in a loop. Use a great deal of care if you have complicated headers or footers in your application that might cause an error if they appear on an error page. Avoid hitting the database or performing any other backend operation that requires either user authorization or that the user’s session be in any specific state. In other words, make sure that the error page is a reliable standalone.

Any status code greater than or equal to 400 increments the ASP.NET Requests Failed performance counter. 401 increments Requests Failed and Requests Not Authorized. 404 and 414 increment both Requests Failed and Requests Not Found. Requests that result in a 500 status code increment Requests Failed and Requests Timed Out. If you’re going to return status codes, you must realize their effects and their implications.

Summary

This chapter examined the debugging tools available to you for creating robust ASP.NET applications. A successful debugging experience includes not only interactive debugging with new features such as datatips, data visualizers, and error notifications, but also powerful options around configurable tracing and logging of information.

Remote debugging is easier than ever with ASP.NET, and the capability to write and debug ASP.NET pages without installing IIS removes yet another layer of complexity from the development process.

Visual Studio and its extensible debugging mechanisms continue to be expanded by intrepid bloggers and enthusiasts, making debugging even less tedious than it has been in the past.

25

File I/O and Streams

Although most of this book concentrates specifically on learning and using the features of ASP.NET 3.5, .NET provides an enormous amount of additional functionality in other areas of the Base Class Library (BCL). This chapter examines a few of the common base classes that you can use to enhance your ASP.NET applications. First, you look at using the frameworks `System.IO` namespace to manage files on the local file system. Next, you explore how to use the various Stream classes within the framework to read from and write different data formats to memory and the local file system. Finally, you learn how to use the .NET Framework to communicate with other computers across the Internet using common protocols such as HTTP and FTP.

A Word about I/O Security

Although this chapter is not specifically about ASP.NET security, you need to understand the impact of local system security on what the ASP.NET Worker Process is allowed to do inside of the IO namespace. Remember that generally, when your code is executed by IIS, it executes under the context of the ASP.NET Worker Process user account (ASPNET) and, therefore, your application may be restricted by that account's security rights. For example, by default, the ASP.NET Worker Process does not have rights to write to the local disk. The two main areas that you should look at to get a very basic understanding of the impact of security on an application are impersonation and user account ACLs. ASP.NET security is discussed thoroughly in Chapter 18.

Additionally, this chapter demonstrates how to use classes in the BCL to delete files and directories and to modify the permissions of directories and files. Recognize that it is entirely possible to permanently delete important data from your hard drive or change the permissions of a resource, which would result in you losing the ability to access the resource. *Be very careful* when using these classes against the file system.

Working with Drives, Directories, and Files

Many times in your ASP.NET applications, you need to interact with the local file system, reading directory structures, reading and writing to files, or performing many other tasks. The `System.IO` namespace within the .NET Framework makes working with file system directories and files very easy. While working with the classes in the `System.IO` namespace, keep in mind that because your ASP.NET applications are executing on the server, the file system you are accessing is the one your Web application is running on. You, of course, cannot use an ASP.NET application to access the end user's file system.

The DriveInfo Class

You can start working with the `System.IO` namespace at the top of the directory tree by using a great new addition to the .NET 3.5 class libraries, the `DriveInfo` class. This class supplements the `GetLogicalDrives()` method of the `Directory` class included in prior versions of the .NET Framework. It provides you with extended information on any drive registered with the server's local file system. You can get information such as the name, type, size, and status of each drive. Listing 25-1 shows you how to create a `DriveInfo` object and display local drive information on a Web page.

Listing 25-1: Displaying local drive information

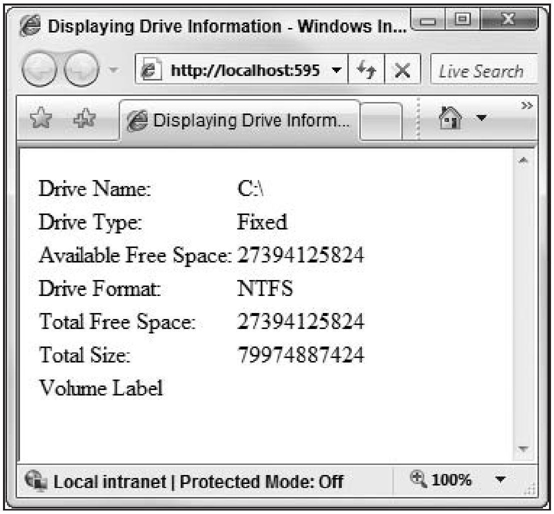
VB

```
<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim drive As New System.IO.DriveInfo("C:\")
        lblDriveName.Text = drive.Name
        lblDriveType.Text = drive.DriveType.ToString()
        lblAvailableFreeSpace.Text = drive.AvailableFreeSpace.ToString()
        lblDriveFormat.Text = drive.DriveFormat
        lblTotalFreeSpace.Text = drive.TotalFreeSpace.ToString()
        lblTotalSize.Text = drive.TotalSize.ToString()
        lblVolumeLabel.Text = drive.VolumeLabel
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Displaying Drive Information</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <table>
            <tr><td>Drive Name:</td><td>
                <asp:Label ID="lblDriveName" runat="server" Text="Label" />
            </td></tr>
            <tr><td>Drive Type:</td><td>
                <asp:Label ID="lblDriveType" runat="server" Text="Label"/>
            </td></tr>
            <tr><td>Available Free Space:</td><td>
               <asp:Label ID="lblAvailableFreeSpace" runat="server" Text="Label" />
            </td></tr>
            <tr><td>Drive Format:</td><td>
                <asp:Label ID="lblDriveFormat" runat="server" Text="Label" />
            </td></tr>
            <tr><td>Total Free Space:</td><td>
                <asp:Label ID="lblTotalFreeSpace" runat="server" Text="Label" />
            </td></tr>
            <tr><td>Total Size:</td><td>
                <asp:Label ID="lblTotalSize" runat="server" Text="Label" />
            </td></tr>
            <tr><td>Volume Label</td><td>
                <asp:Label ID="lblVolumeLabel" runat="server" Text="Label" />
            </td></tr>
        </table>
    </div>
    </form>
</body>
</html>
```

C#

```
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
     {
        System.IO.DriveInfo drive = new System.IO.DriveInfo(@"C:\");
        lblDriveName.Text = drive.Name;
        lblDriveType.Text = drive.DriveType.ToString();
        lblAvailableFreeSpace.Text = drive.AvailableFreeSpace.ToString();
        lblDriveFormat.Text = drive.DriveFormat;
        lblTotalFreeSpace.Text = drive.TotalFreeSpace.ToString();
        lblTotalSize.Text = drive.TotalSize.ToString();
        lblVolumeLabel.Text = drive.VolumeLabel;
    }
</script>
```

One of the more interesting properties in the sample is the `DriveType` enumeration. This read-only enumeration tells you what the drive type is, for example CD-ROM, Fixed, Ram, or Removable. Figure 25-1 shows you what the page looks like when you view it in a browser.


Displaying Drive Information - Windows In...
http://localhost:595
Live Search
Displaying Drive Inform...
Local intranet | Protected Mode: Off
100%

Drive Name:	C:\
Drive Type:	Fixed
Available Free Space:	27394125824
Drive Format:	NTFS
Total Free Space:	27394125824
Total Size:	79974887424
Volume Label	

Figure 25-1

You can also enumerate through all the drives on the local file system by using the `DriveInfo`'s static `GetDrives()` method. Listing 25-2 shows an example of enumerating through the local file system drives and adding each drive as a root node to a TreeView control.

Listing 25-2: Enumerating through local file system drives

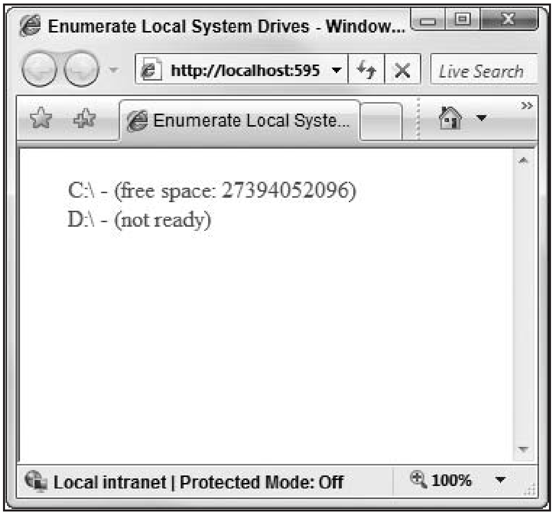
VB

```
<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        If (Not Page.IsPostBack) Then

        For Each drive As System.IO.DriveInfo In System.IO.DriveInfo.GetDrives()

            Dim node As TreeNode = New TreeNode()
            node.Value = drive.Name

            ' Make sure the drive is ready before we access it
            If (drive.IsReady) Then
                node.Text = drive.Name & _
                            " - (free space: " & drive.AvailableFreeSpace & ")"
            Else
                node.Text = drive.Name & " - (not ready)"
            End If

            Me.TreeView1.Nodes.Add(node)
        Next

        End If
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Enumerate Local System Drives</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <table>
            <tr>
                <td style="width: 100px" valign="top">
                    <asp:TreeView ID="TreeView1" runat="server"></asp:TreeView>
                </td>
            </tr>
        </table>
    </div>
    </form>
</body>
</html>
```

C#

```
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
            if (!Page.IsPostBack)
            {

            foreach (System.IO.DriveInfo drive in System.IO.DriveInfo.GetDrives())
            {
                TreeNode node = new TreeNode();
                node.Value = drive.Name;

                //Make sure the drive is ready before we access it
                if (drive.IsReady)
                    node.Text = drive.Name +
                                " - (free space: " + drive.AvailableFreeSpace + ")";
                else
                    node.Text = drive.Name + " - (not ready)";

                this.TreeView1.Nodes.Add(node);
            }

            }
        }
    </script>
```

Notice that, in this sample, the drive object's `IsReady` property is a read-only property used to test whether the drive is accessible. If you are enumerating drives, it's always a good idea to test for this before attempting to access any of the other drive properties because removable drives and network drives may not always be available when your code is executed. Figure 25-2 shows what the page looks like when viewed in the browser.


Enumerate Local System Drives - Window...
http://localhost:595
Live Search
Enumerate Local Syste...
Local intranet | Protected Mode: Off
100%

C:\ - (free space: 27394052096)
D:\ - (not ready)

Figure 25-2

The Directory and DirectoryInfo Classes

Next, you can build on the previous examples and add the capability to browse through the system's directory structure. The `System.IO` namespace contains two classes for working with file system directories, the `Directory` and `DirectoryInfo` classes. The `Directory` class exposes static methods you can use to create, move, and delete directories. The `DirectoryInfo` represents a specific directory and lets you perform many of the same actions as the `Directory` class on the specific directory. Additionally, it enumerates child directories and files.

To continue the example, you can use the `GetDirectories()` method of the `DirectoryInfo` class to create a recursive method that loops through each system drive directory tree and adds the directories to a TreeView control to create a small directory browser. Listing 25-3 shows how to create a recursive `LoadDirectories()` method to walk through the local file system's directory structure.

Listing 25-3: Enumerating file system directories

VB

```
<script runat="server">

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)

        If (Not Page.IsPostBack) Then

        For Each drive As System.IO.DriveInfo In System.IO.DriveInfo.GetDrives()

            Dim node As TreeNode = New TreeNode()
            node.Value = drive.Name

            If (drive.IsReady) Then
                node.Text = drive.Name & _
                            " - (free space: " & drive.AvailableFreeSpace & ")"

                LoadDirectories(node, drive.Name)
            Else
                node.Text = drive.Name & " - (not ready)"
            End If

            Me.TreeView1.Nodes.Add(node)
        Next

        End If

        Me.TreeView1.CollapseAll()

    End Sub

    Private Sub LoadDirectories(ByVal parent As TreeNode, ByVal path As String)

        Dim directory As System.IO.DirectoryInfo = _
                  New System.IO.DirectoryInfo(path)

        Try
            For Each d As System.IO.DirectoryInfo In directory.GetDirectories()

                Dim node As TreeNode = New TreeNode(d.Name, d.FullName)

                parent.ChildNodes.Add(node)
```

Continued

```
                'Recurse the current directory
                LoadDirectories(node, d.FullName)
            Next
        Catch ex As System.UnauthorizedAccessException
            parent.Text += " (Access Denied)"
        Catch ex As System.IO.IOException
            parent.Text += " (Unknown Error: " + ex.Message + ")"
        End Try
    End Sub
</script>
```

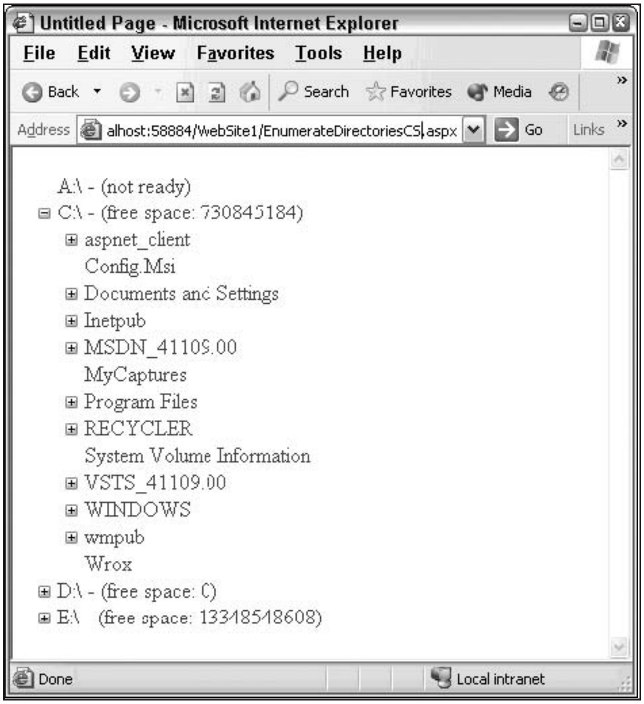
C#

```
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {

        foreach (System.IO.DriveInfo drive in System.IO.DriveInfo.GetDrives())
        {
            TreeNode node = new TreeNode();
            node.Value = drive.Name;

            if (drive.IsReady)
            {
                node.Text = drive.Name +
                            " - (free space: " + drive.AvailableFreeSpace + ")";

                LoadDirectories(node, drive.Name);
            }
            else
                node.Text = drive.Name + " - (not ready)";

            this.TreeView1.Nodes.Add(node);
        }

        }

        this.TreeView1.CollapseAll();
    }

    private void LoadDirectories(TreeNode parent, string path)
    {
        System.IO.DirectoryInfo directory = new System.IO.DirectoryInfo(path);

        try
        {
            foreach (System.IO.DirectoryInfo d in directory.GetDirectories())
            {
                TreeNode node = new TreeNode(d.Name, d.FullName);

                parent.ChildNodes.Add(node);
```

Continued

```
                //Recurs the current directory
                LoadDirectories(node, d.FullName);
            }
        }
        catch (System.UnauthorizedAccessException e)
        {
            parent.Text += " (Access Denied)";
        }
        catch (System.IO.IOException e)
        {
            parent.Text += " (Unknown Error: " + e.Message + ")";
        }
    }
</script>
```

Figure 25-3 shows what the page should look like in the browser. You should now be able to browse the directory tree, much as you do in Windows Explorer, by opening and closing the TreeView nodes.


Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address
alhost:58884/WebSite1/EnumerateDirectoriesCS.aspx
Go
Links
Done
Local intranet

```
  A:\ - (not ready)
⊟ C:\ - (free space: 730845184)
    ⊞ aspnet_client
      Config.Msi
    ⊞ Documents and Settings
    ⊞ Inetpub
    ⊞ MSDN_41109.00
      MyCaptures
    ⊞ Program Files
    ⊞ RECYCLER
      System Volume Information
    ⊞ VSTS_41109.00
    ⊞ WINDOWS
    ⊞ wmpub
      Wrox
⊞ D:\ - (free space: 0)
⊞ E:\   (free space: 13348548608)
```

Figure 25-3

Notice that the example continuously creates new instances of the `DirectoryInfo` class each time the method executes in order to continue to enumerate the directory tree. You could also extend this example by displaying some additional properties as part of the `Node` text, such as the `CreationTime` or `Attributes`.

To perform only a specific action, you don't have to create an instance of the `DirectoryInfo` class. You can simply use the static methods exposed by the `Directory` class. These methods allow you to create, read properties from, and delete a directory. Rather than creating an object instance that represents a specific path and exposes methods that act on that path, the static methods exposed by the `Directory` class generally require you to pass the path as a method parameter. Listing 25-4 shows how you can use the static methods exposed by the `Directory` class to create, read properties from, and delete a directory.

Remember to be very careful when deleting a folder from your hard drive. It is possible to permanently delete important data from your system or change the permissions of a resource, which would result in your losing the ability to access the resource.

Listing 25-4: Working with the static methods of the Directory class

VB

```
<%@ Import Namespace="System.IO" %>

<script runat="server">

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)

        Directory.CreateDirectory("C:\Wrox")

        If Directory.Exists("C:\Wrox") Then

            Me.Label1.Text = _
                    Directory.GetCreationTime("C:\Wrox").ToString()
            Me.Label2.Text = _
                    Directory.GetLastAccessTime("C:\Wrox").ToString()
            Me.Label3.Text = _
                   Directory.GetLastWriteTime("C:\Wrox").ToString()

            Directory.Delete("C:\Wrox")
        End If
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Using Static Methods</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        Creation Time:
             <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label><br />
        Last Access Time:
             <asp:Label ID="Label2" runat="server" Text="Label"></asp:Label><br />
        Last Write Time:
              <asp:Label ID="Label3" runat="server" Text="Label"></asp:Label>
    </div>
    </form>
</body>
</html>
```

Continued

C#

```
<%@ Import Namespace="System.IO" %>

<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        Directory.CreateDirectory(@"C:\Wrox");

        if (Directory.Exists(@"C:\Wrox") )
        {
            this.Label1.Text =
                     Directory.GetCreationTime(@"C:\Wrox").ToString();
            this.Label2.Text =
                     Directory.GetLastAccessTime(@"C:\Wrox").ToString();
            this.Label3.Text =
                     Directory.GetLastWriteTime(@"C:\Wrox").ToString();

            Directory.Delete(@"C:\Wrox");
        }
    }
</script>
```

When you load this page in the browser, you will see that the `Creation Time`, `Last Access Time`, and `Last Write Time` are displayed. Additionally, if you open Windows Explorer, you will see that the Wrox directory has been deleted.

Using Relative Paths and Setting and Getting the Current Directory

When an ASP.NET page is executed, the thread used to execute the code that generates the page, by default, has a current working directory. It uses this directory as its base directory if you have specified relative paths in your application. Therefore, if you pass a relative filename into any `System.IO` class, the file is assumed to be located in the current working directory.

For example, the default working directory for the ASP.NET Development Server is a directory under your Visual Studio install root. If you installed Visual Studio in `C:\Program Files`, your ASP.NET Development Server working directory would be `c:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE`.

You can find the location of your working directory by using the `Directory` class's `GetCurrentDirectory()` method. In addition, you can change the current working directory using the `Directory` class's `SetCurrentDirectory()` method.

Listing 25-5 shows you how to set and then display your working directory.

Listing 25-5: Setting and displaying the application's working directory

VB

```
<%@ Import Namespace="System.IO" %>

<script runat="server">
```

```
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Me.Label1.Text = Directory.GetCurrentDirectory()
        Directory.SetCurrentDirectory("C:\Wrox")
        Me.Label2.Text = Directory.GetCurrentDirectory()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Set and Display the Working Directory</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        Old Working Directory:
            <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label><br />
        New Working Directory:
            <asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
    </div>
    </form>
</body>
</html>
```

C#

```
<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        this.Label1.Text = Directory.GetCurrentDirectory();
        Directory.SetCurrentDirectory(@"C:\Wrox");
        this.Label2.Text = Directory.GetCurrentDirectory();
    }
</script>
```

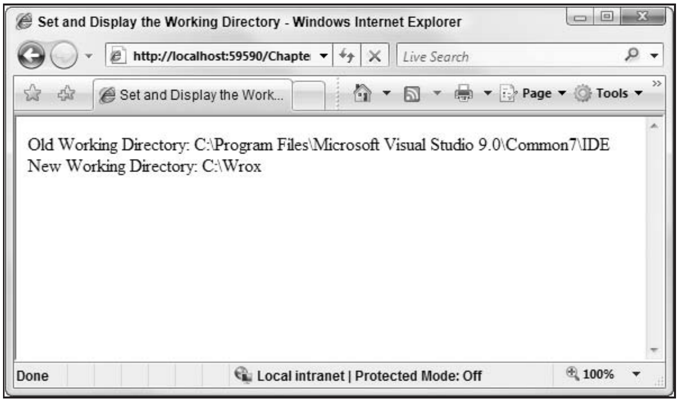
Note that the directory parameter you specify in the `SetCurrentDirectory()` method must already exist; otherwise, ASP.NET throws an exception. Knowing this, it would probably be a good idea to use the `Exists()` method of the `Directory` class to make sure the directory you are specifying does, in fact, already exist before you try to change the working directory.

When you execute this code, you should see that it displays the original working directory, and then displays the new working directory after you change it. Figure 25-4 shows what the page looks like when executed.

File and FileInfo

Now that you can effectively display and browse a directory tree, you can expand the example even further by displaying the files located in the directory that is currently selected in your TreeView control.

The simplest way to display the files is to bind a `FileInfo` array to a `GridView`. This example uses the `GetFiles()` method of the `DirectoryInfo` class because it returns an array of `FileInfo` objects. You want to use this method because the `FileInfo` object enables you to display some properties of each file. (If you want to display only the filenames, you could use the `Directory` class's `GetFiles()` method, which returns a simple string array of filenames.)


Set and Display the Working Directory - Windows Internet Explorer
http://localhost:59590/Chapte
Live Search
Set and Display the Work...
Page
Tools
Done
Local intranet | Protected Mode: Off
100%

Old Working Directory: C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE
New Working Directory: C:\Wrox

Figure 25-4

Listing 25-6 shows how to use the TreeView control's `SelectedNodeChanged` event to bind your `GridView` with the file information.

Listing 25-6: Binding a GridView to directory files

VB

```
<script runat="server">

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)

        If (Not Page.IsPostBack) Then

        For Each drive As System.IO.DriveInfo In System.IO.DriveInfo.GetDrives()

            Dim node As TreeNode = New TreeNode()
            node.Value = drive.Name

            If (drive.IsReady) Then
                node.Text = drive.Name & _
                            " - (free space: " & drive.AvailableFreeSpace & ")"

                LoadDirectories(node, drive.Name)
            Else
                node.Text = drive.Name & " - (not ready)"
            End If

            Me.TreeView1.Nodes.Add(node)
        Next

        End If

        Me.TreeView1.CollapseAll()
```

Continued

```
    End Sub

    Private Sub LoadDirectories(ByVal parent As TreeNode, ByVal path As String)

        Dim directory As System.IO.DirectoryInfo = _
                  New System.IO.DirectoryInfo(path)

        Try
            For Each d As System.IO.DirectoryInfo In directory.GetDirectories()

                Dim node As TreeNode = New TreeNode(d.Name, d.FullName)

                parent.ChildNodes.Add(node)

                'Recurse the current directory
                LoadDirectories(node, d.FullName)
            Next
        Catch ex As System.UnauthorizedAccessException
            parent.Text += " (Access Denied)"
        Catch ex As Exception
            parent.Text += " (Unknown Error: " + ex.Message + ")"
        End Try
    End Sub

    Protected Sub TreeView1_SelectedNodeChanged _
        (ByVal sender As Object, ByVal e As System.EventArgs)

        Dim directory As System.IO.DirectoryInfo = _
                       New System.IO.DirectoryInfo(Me.TreeView1.SelectedNode.Value)

        Me.GridView1.DataSource = directory.GetFiles()
        Me.GridView1.DataBind()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Binding a Gridview </title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <table>
            <tr>
                <td style="width: 100px" valign="top">
                   <asp:TreeView ID="TreeView1" runat="server"
                         OnSelectedNodeChanged="TreeView1_SelectedNodeChanged">
                   </asp:TreeView>
                </td>
                <td valign=top>
                    <asp:GridView ID="GridView1" runat="server"
                         AutoGenerateColumns=False GridLines=None CellPadding=3>
```

Continued

```
                        <Columns>
                            <asp:BoundField DataField="Name" HeaderText="Name"
                                 HeaderStyle-HorizontalAlign=Left
                                 HeaderStyle-Font-Bold=true />
                            <asp:BoundField DataField="Length" HeaderText="Size"
                                 ItemStyle-HorizontalAlign=Right
                                 HeaderStyle-HorizontalAlign=Right
                                 HeaderStyle-Font-Bold=true />
                            <asp:BoundField DataField="LastWriteTime"
                                 HeaderText="Date Modified"
                                 HeaderStyle-HorizontalAlign=Left
                                 HeaderStyle-Font-Bold=true />
                        </Columns>
                    </asp:GridView>
                </td>
            </tr>
        </table>
    </div>
    </form>
</body>
</html>
```

C#

```
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {

        if (!Page.IsPostBack)
        {

        foreach (System.IO.DriveInfo drive in System.IO.DriveInfo.GetDrives())
        {
            TreeNode node = new TreeNode();
            node.Value = drive.Name;

            // Make sure the drive is ready before we access it
            if (drive.IsReady)
            {
                node.Text = drive.Name +
                            " - (free space: " + drive.AvailableFreeSpace + ")";

                LoadDirectories(node, drive.Name);
            }
            else
                node.Text = drive.Name + " - (not ready)";

            this.TreeView1.Nodes.Add(node);
        }

        }

        this.TreeView1.CollapseAll();
    }
```

Continued

```
    private void LoadDirectories(TreeNode parent, string path)
    {
        System.IO.DirectoryInfo directory = new System.IO.DirectoryInfo(path);

        try
        {
            foreach (System.IO.DirectoryInfo d in directory.GetDirectories())
            {
                TreeNode node = new TreeNode(d.Name, d.FullName);

                parent.ChildNodes.Add(node);

                //Recurse the current directory
                LoadDirectories(node, d.FullName);
            }
        }
        catch (System.UnauthorizedAccessException e)
        {
            parent.Text += " (Access Denied)";
        }
        catch (Exception e)
        {
            parent.Text += " (Unknown Error: " + e.Message + ")";
        }
    }

    protected void TreeView1_SelectedNodeChanged(object sender, EventArgs e)
    {
        System.IO.DirectoryInfo directory =
                  new System.IO.DirectoryInfo(this.TreeView1.SelectedNode.Value);

        this.GridView1.DataSource = directory.GetFiles();
        this.GridView1.DataBind();
    }
</script>
```

Figure 25-5 shows what your Web page looks like after you have selected a directory and your grid has been bound to the `FileInfo` array.

Keep in mind that, as in the Load Directory example, you can also enumerate though the `FileInfo` array to display the information. Listing 25-7 shows you how to enumerate through the `FileInfo` array and display the properties to the page.

Listing 25-7: Manually enumerating directory files

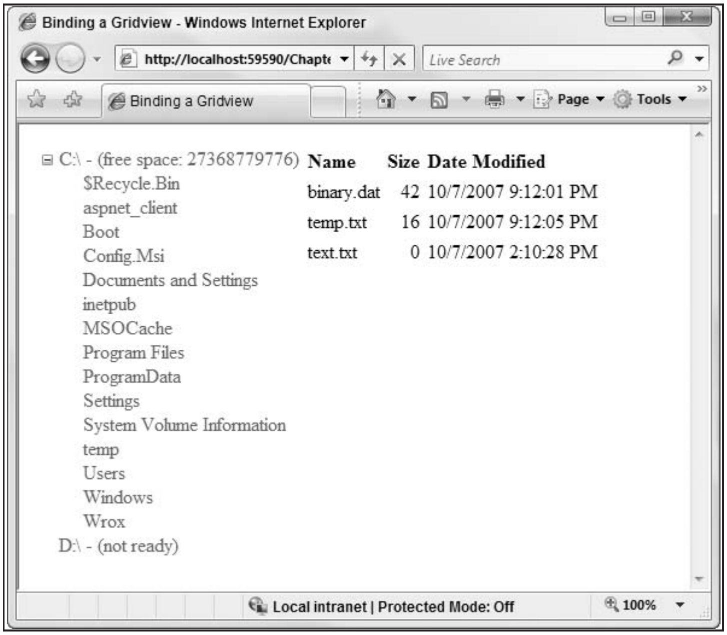
VB

```
Dim dir as New System.IO.DirectoryInfo("C:\")
For Each file as System.IO.FileInfo In dir.GetFiles("*.*")
     Response.Write(file.Name & "<BR>")
     Response.Write(file.LastWriteTime.ToString() & "<BR>")
     Response.Write(file.Attributes.ToString() & "<BR>")
Next
```

Continued

C#

```
System.IO.DirectoryInfo dir = new System.IO.DirectoryInfo(@"C:\");
foreach (System.IO.FileInfo file in dir.GetFiles("*.*"))
{
     Response.Write(file.Name + "<BR>");
     Response.Write(file.LastWriteTime.ToString() + "<BR>");
     Response.Write(file.Attributes.ToString() + "<BR>");
}
```


Binding a Gridview - Windows Internet Explorer
http://localhost:59590/Chapte
Live Search
Binding a Gridview
Page
Tools
Local intranet | Protected Mode: Off
100%

- C:\ - (free space: 27368779776)
  - $Recycle.Bin
  - aspnet_client
  - Boot
  - Config.Msi
  - Documents and Settings
  - inetpub
  - MSOCache
  - Program Files
  - ProgramData
  - Settings
  - System Volume Information
  - temp
  - Users
  - Windows
  - Wrox
- D:\ - (not ready)

Name	Size	Date Modified
binary.dat	42	10/7/2007 9:12:01 PM
temp.txt	16	10/7/2007 9:12:05 PM
text.txt	0	10/7/2007 2:10:28 PM

Figure 25-5

Listing 25-7 also shows that you can provide a file filter to the `GetFiles()` method. This allows you to limit the results from the method to specific file extensions or to files matching a specific filename part.

Working with Paths

Although working with files and directories has been pretty easy, even going all the way back to good old ASP, one of the most problematic areas has always been working with paths. Many lines of code have been written by developers to deal with concatenating partial paths together, making sure files have extensions, evaluating those extensions, stripping filenames off of paths, and even more.

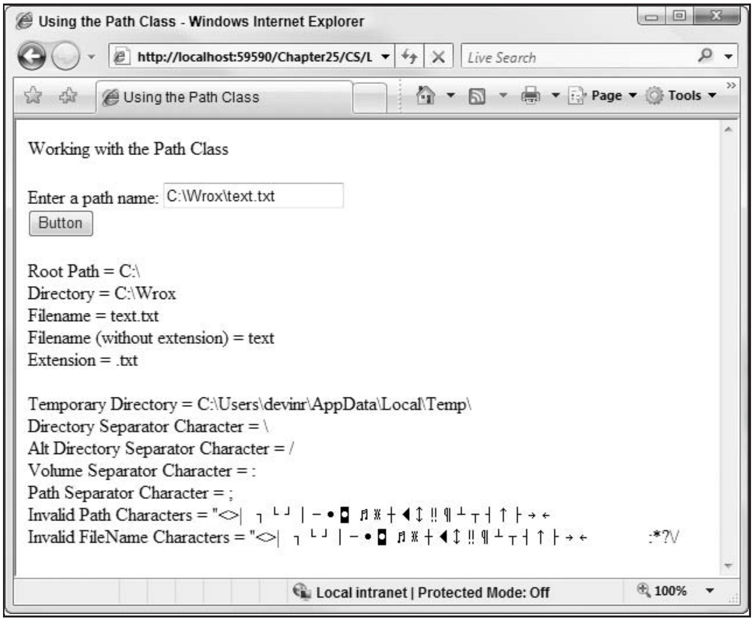
Thankfully, the .NET Framework provides you with a class just for dealing with paths. The `System.IO.Path` class exposes a handful of static methods that make dealing with paths a snap. The following table lists the static methods exposed by the `Path` class.

Method	**Description**
`ChangeExtension`	Changes the extension of the provided path string to the provided new extension.
`Combine`	Returns a single combined path from two partial path strings.
`GetDirectoryName`	Returns the directory or directories of the provided path.
`GetExtension`	Returns the extension of the provided path.
`GetFileName`	Returns the filename of the provided path.
`GetFileNameWithoutExtension`	Returns the filename without its extension of the provided path.
`GetFullPath`	Given a non-rooted path, returns a rooted pathname based on the current working directory. For example, if the path passed in is `"temp"` and the current working directory is `c:\MyWebsite`, the method returns `C:\MyWebsite\temp`.
`GetInvalidFileNameChars`	Returns an array of characters that are not allowed in filenames for the current system.
`GetInvalidPathChars`	Returns an array of characters that are not allowed in pathnames for the current system.
`GetPathRoot`	Returns the root path.
`GetTempFileName`	Returns a temporary filename, located in the temporary directory returned by `GetTempPath`.
`GetTempPath`	Returns the temporary directory name.
`HasExtension`	Returns a Boolean value indicating whether a path has an extension.
`IsPathRooted`	Returns a Boolean indicating if a path is rooted.

As an example of using the `Path` class, the application shown in Figure 25-6 lets you enter a path and then displays the component parts of the path such as the root path (logical drive), the directory, filename, and extension.

> *The* `GetInvalidPathChars` *and* `GetInvalidFileNameChars` *methods return an array of characters that are not allowed in path and filenames, respectively. Although the specific invalid characters are dependent on the platform the application is running on, the arrays returned by these methods will most likely contain elements such as non-printable characters, special Unicode characters, or characters from non-Latin–based character sets. The characters that your browser is capable of rendering will depend on your specific platform setup. Characters that your browser is incapable of rendering properly will display as the generic square box shown in Figure 25-6.*

The code in Listing 25-8 shows how the various methods and constant properties of the `Path` class have been used to create the application shown in Figure 25-6.


Using the Path Class - Windows Internet Explorer
http://localhost:59590/Chapter25/CS/L
Live Search
Using the Path Class
Page
Tools
Button
Local intranet | Protected Mode: Off
100%

Working with the Path Class

Enter a path name: C:\Wrox\text.txt

Root Path = C:\
Directory = C:\Wrox
Filename = text.txt
Filename (without extension) = text
Extension = .txt

Temporary Directory = C:\Users\devinr\AppData\Local\Temp\
Directory Separator Character = \
Alt Directory Separator Character = /
Volume Separator Character = :
Path Separator Character = ;
Invalid Path Characters = "<>| ┐ └ ┘ | ‒ • ◘ ♬ ※ ┼ ◄ ↕ ‼ ¶ ┴ ┬ ┤ ↑ ├ → ←
Invalid FileName Characters = "<>| ┐ └ ┘ | ‒ • ◘ ♬ ※ ┼ ◄ ↕ ‼ ¶ ┴ ┬ ┤ ↑ ├ → ← :*?\/

Figure 25-6

Listing 25-8: Using the Path class

VB

```
<%@ Import Namespace="System.IO" %>

<script runat="server">

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        If Page.IsPostBack Then
            Me.lblRootPath.Text = Path.GetPathRoot(Me.txtPathName.Text)
            Me.lblDirectoryName.Text = Path.GetDirectoryName(Me.txtPathName.Text)
            Me.lblFileName.Text = Path.GetFileName(Me.txtPathName.Text)
            Me.lblFileNameWithoutExtension.Text = _
                         Path.GetFileNameWithoutExtension(Me.txtPathName.Text)
            Me.lblExtension.Text = Path.GetExtension(Me.txtPathName.Text)

            Me.lblTemporaryPath.Text = Path.GetTempPath()
            Me.lblDirectorySeparatorChar.Text = _
                         Path.DirectorySeparatorChar.ToString()
            Me.lblAltDirectorySeparatorChar.Text = _
                         Path.AltDirectorySeparatorChar.ToString()
            Me.lblVolumeSeparatorChar.Text = Path.VolumeSeparatorChar.ToString()
            Me.lblPathSeparator.Text = Path.PathSeparator.ToString()

            Me.lblInvalidChars.Text = _
                 HttpUtility.HtmlEncode(New String(Path.GetInvalidPathChars()))
```

Continued

```
                Me.lblInvalidFileNameChars.Text = _
                     HttpUtility.HtmlEncode(New String(Path.GetInvalidFileNameChars()))
            End If
        End Sub
    </script>

    <html xmlns="http://www.w3.org/1999/xhtml" >
    <head runat="server">
        <title>Using the Path Class</title>
    </head>
    <body>
        <form id="form1" runat="server">
        <div>
            Working with the Path Class<br />
            <br />
            Enter a path name:
            <asp:TextBox ID="txtPathName" runat="server"></asp:TextBox><br />
            <asp:Button ID="Button1" runat="server" Text="Button" /><br />
            <br />
            Root Path =
            <asp:Label ID="lblRootPath" runat="server" Text="Label" />
            <br />
            Directory =
            <asp:Label ID="lblDirectoryName" runat="server" Text="Label" />
            <br />
            Filename =
            <asp:Label ID="lblFileName" runat="server" Text="Label" />
            <br />
            Filename (without extension) =
            <asp:Label ID="lblFileNameWithoutExtension" runat="server" Text="Label" />
            <br />
            Extension =
            <asp:Label ID="lblExtension" runat="server" Text="Label" />
            <br />
            <br />
            Temporary Directory =
            <asp:Label ID="lblTemporaryPath" runat="server" Text="Label" />
            <br />
            Directory Separator Character =
            <asp:Label ID="lblDirectorySeparatorChar" runat="server" Text="Label" />
            <br />
            Alt Directory Separator Character =
            <asp:Label ID="lblAltDirectorySeparatorChar" runat="server" Text="Label" />
            <br />
            Volume Separator Character =
            <asp:Label ID="lblVolumeSeparatorChar" runat="server" Text="Label" />
            <br />
            Path Separator Character =
            <asp:Label ID="lblPathSeparator" runat="server" Text="Label" />
            <br />
            Invalid Path Characters =
            <asp:Label ID="lblInvalidChars" runat="server" Text="Label" />
            <br />
            Invalid FileName Characters =
```

Continued

```
        <asp:Label ID="lblInvalidFileNameChars" runat="server" Text="Label" />

    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Import Namespace="System.IO" %>

<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        if (Page.IsPostBack)
        {
            this.lblRootPath.Text =
                           Path.GetPathRoot(this.txtPathName.Text);
            this.lblDirectoryName.Text =
                           Path.GetDirectoryName(this.txtPathName.Text);
            this.lblFileName.Text =
                           Path.GetFileName(this.txtPathName.Text);
            this.lblFileNameWithoutExtension.Text =
                           Path.GetFileNameWithoutExtension(this.txtPathName.Text);
            this.lblExtension.Text =
                           Path.GetExtension(this.txtPathName.Text);

            this.lblTemporaryPath.Text = Path.GetTempPath();
            this.lblDirectorySeparatorChar.Text =
                           Path.DirectorySeparatorChar.ToString();
            this.lblAltDirectorySeparatorChar.Text =
                           Path.AltDirectorySeparatorChar.ToString();
            this.lblVolumeSeparatorChar.Text = Path.VolumeSeparatorChar.ToString();
            this.lblPathSeparator.Text = Path.PathSeparator.ToString();

            this.lblInvalidChars.Text =
               HttpUtility.HtmlEncode( new String(Path.GetInvalidPathChars() ) );
            this.lblInvalidFileNameChars.Text =
               HttpUtility.HtmlEncode( new String(Path.GetInvalidFileNameChars()));
        }
    }
</script>
```

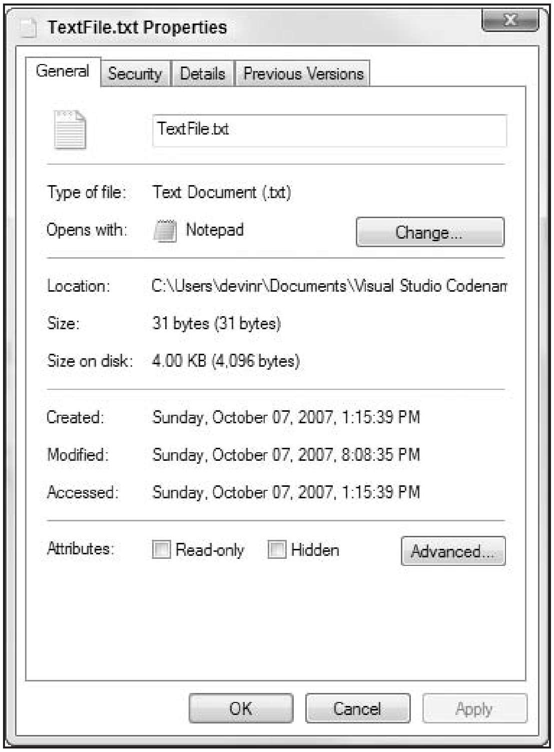
File and Directory Properties, Attributes, and Access Control Lists

Finally, this section explains how you can access and modify file and directory properties, attributes, and Access Control Lists.

> *Samples in this section use a simple text file called* `TextFile.txt` *to demonstrate the concepts. You can either create this file or substitute your own file in the sample code. The samples assume the file has been added to the Web site and use the* `Server.MapPath` *method to determine the full filepath.*

Properties and Attributes

Files and directories share certain properties that you can use to determine the age of a file or directory, when it was last modified, and what attributes have been applied. These properties can be viewed by opening the file's Properties dialog. You can open this dialog from Windows Explorer by either right-clicking on the file and selecting Properties from the context menu, or selecting Properties from the File menu. Figure 25-7 shows the file's Properties window for the text document.



TextFile.txt

Type of file: Text Document (.txt)

Opens with: Notepad

Location: C:\Users\devinr\Documents\Visual Studio Codenam

Size: 31 bytes (31 bytes)

Size on disk: 4.00 KB (4,096 bytes)

Created: Sunday, October 07, 2007, 1:15:39 PM

Modified: Sunday, October 07, 2007, 8:08:35 PM

Accessed: Sunday, October 07, 2007, 1:15:39 PM

Attributes: ☐ Read-only ☐ Hidden

Figure 25-7

Both the `DirectoryInfo` and the `FileInfo` classes let you access these properties and modify them. Listing 25-9 shows you an example of displaying the file properties.

Listing 25-9: Displaying and modifying the file properties

VB

```
Dim file As New System.IO.FileInfo(Server.MapPath("TextFile.txt"))
Response.Write("Location: " & file.FullName & "<BR>")
Response.Write("Size: " & file.Length & "<BR>")
Response.Write("Created: " & file.CreationTime & "<BR>")
Response.Write("Modified: " & file.LastWriteTime & "<BR>")
Response.Write("Accessed: " & file.LastAccessTime & "<BR>")
Response.Write("Attributes: " & file.Attributes)
```

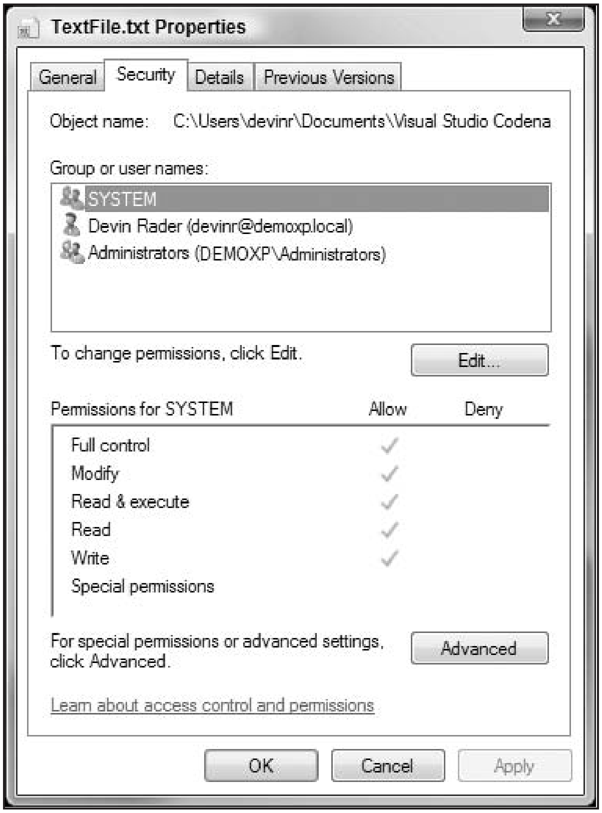
Continued

C#

```
System.IO.FileInfo file = new System.IO.FileInfo(Server.MapPath("TextFile.txt"));
Response.Write("Location: " + file.FullName + "<BR>");
Response.Write("Size: " + file.Length + "<BR>");
Response.Write("Created: " + file.CreationTime + "<BR>");
Response.Write("Modified: " + file.LastWriteTime + "<BR>");
Response.Write("Accessed: " + file.LastAccessTime + "<BR>");
Response.Write("Attributes: " + file.Attributes);
```

Access Control Lists

Although getting the properties and attributes is useful, what many developers need is the capability to actually change the Access Control Lists, or ACLs — pronounced *Ackels* — on directories and files. ACLs are the way resources such as directories and files are secured in the NTFS file system, which is the file system used by Windows XP, NT 4.0, 2000, and 2003. You can view a file's ACLs by selecting the Security tab from the file's Properties dialog. Figure 25-8 shows the ACLs set for the `TextFile.txt` file you created.



Object name: C:\Users\devinr\Documents\Visual Studio Codena

Group or user names:
SYSTEM
Devin Rader (devinr@demoxp.local)
Administrators (DEMOXP\Administrators)

To change permissions, click Edit.

Permissions for SYSTEM	Allow	Deny
Full control	✓	
Modify	✓	
Read & execute	✓	
Read	✓	
Write	✓	
Special permissions		

For special permissions or advanced settings, click Advanced.

Figure 25-8

Using the new `System.AccessControl` namespace in the .NET Framework, you can query the file system for the ACL information and display it in a Web page, as shown in Listing 25-10.

Listing 25-10: Access Control List information

VB

```
<script runat="server">

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        ' retrieve the AccessControl information for this file
        Dim sec As System.Security.AccessControl.FileSecurity
        sec = System.IO.File.GetAccessControl(Server.MapPath("TextFile.txt"))

        Me.Label1.Text = _
                 sec.GetOwner(GetType(System.Security.Principal.NTAccount)).Value

        ' retrieve the collection of access rules
        Dim auth As System.Security.AccessControl.AuthorizationRuleCollection = _
                             sec.GetAccessRules(true, true, _
                             GetType(System.Security.Principal.NTAccount))

        Dim tc As TableCell
        ' loop through the rule collection and add a table row for each rule
        For Each r As System.Security.AccessControl.FileSystemAccessRule In auth

            Dim tr As New TableRow()

            tc = New TableCell()
            tc.Text = r.AccessControlType.ToString() ' deny or allow
            tr.Cells.Add(tc)

            tc = New TableCell()
            tc.Text = r.IdentityReference.Value ' who
            tr.Cells.Add(tc)

            tc = New TableCell()
            tc.Text = r.InheritanceFlags.ToString()
            tr.Cells.Add(tc)

            tc = New TableCell()
            tc.Text = r.IsInherited.ToString()
            tr.Cells.Add(tc)

            tc = New TableCell()
            tc.Text = r.PropagationFlags.ToString()
            tr.Cells.Add(tc)

            tc = New TableCell()
            tc.Text = r.FileSystemRights.ToString()
            tr.Cells.Add(tc)

            Table1.Rows.Add(tr)
        Next
    End Sub
</script>
```

Continued

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Displaying ACL Information</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <p><b>File Owner:</b>
              <asp:Label ID="Label1" runat="server" Text="Label" /></p>
        <p>
        Access Rules:<br />
        <asp:Table ID="Table1" runat="server" CellPadding="2" GridLines="Both">
            <asp:TableRow>
                <asp:TableHeaderCell>Control Type</asp:TableHeaderCell>
                <asp:TableHeaderCell>Identity</asp:TableHeaderCell>
                <asp:TableHeaderCell>Inheritance Flags</asp:TableHeaderCell>
                <asp:TableHeaderCell>Is Inherited</asp:TableHeaderCell>
                <asp:TableHeaderCell>Propagation Flags</asp:TableHeaderCell>
                <asp:TableHeaderCell>File System Rights</asp:TableHeaderCell>
            </asp:TableRow>
        </asp:Table>
        </p>
    </div>
    </form>
</body>
</html>
```

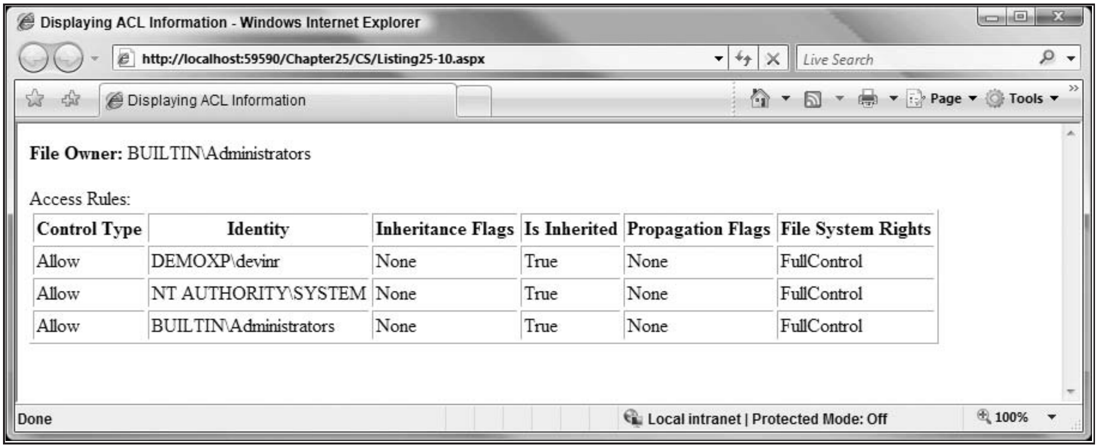
C#

```
<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        // retrieve the AccessControl information for this file
        System.Security.AccessControl.FileSecurity sec =
                 System.IO.File.GetAccessControl(Server.MapPath("TextFile.txt"));

        this.Label1.Text =
                 sec.GetOwner( typeof(System.Security.Principal.NTAccount) ).Value;

        // retrieve the collection of access rules
        System.Security.AccessControl.AuthorizationRuleCollection auth =
                                    sec.GetAccessRules(true, true,
                                    typeof (System.Security.Principal.NTAccount));

        TableCell tc;
        // loop through the rule collection and add a table row for each rule
        foreach (System.Security.AccessControl.FileSystemAccessRule r in auth)
        {
            TableRow tr = new TableRow();

            tc = new TableCell();
            tc.Text = r.AccessControlType.ToString(); // deny or allow
            tr.Cells.Add(tc);
```

Continued

```
            tc = new TableCell();
            tc.Text = r.IdentityReference.Value; // who
            tr.Cells.Add(tc);

            tc = new TableCell();
            tc.Text = r.InheritanceFlags.ToString();
            tr.Cells.Add(tc);

            tc = new TableCell();
            tc.Text = r.IsInherited.ToString();
            tr.Cells.Add(tc);

            tc = new TableCell();
            tc.Text = r.PropagationFlags.ToString();
            tr.Cells.Add(tc);

            tc = new TableCell();
            tc.Text = r.FileSystemRights.ToString();
            tr.Cells.Add(tc);

            Table1.Rows.Add(tr);
        }
    }
</script>
```

Figure 25-9 shows what the page looks like when it is executed. Note that the Identity column might be different depending on whom you are logged in as when you run the page and what security mode the application is running under (Integrated Windows Authentication, Basic, or Anonymous).


Displaying ACL Information - Windows Internet Explorer
http://localhost:59590/Chapter25/CS/Listing25-10.aspx
Live Search
Displaying ACL Information
Page
Tools
Done
Local intranet | Protected Mode: Off
100%

File Owner: BUILTIN\Administrators

Access Rules:

Control Type	Identity	Inheritance Flags	Is Inherited	Propagation Flags	File System Rights
Allow	DEMOXP\devinr	None	True	None	FullControl
Allow	NT AUTHORITY\SYSTEM	None	True	None	FullControl
Allow	BUILTIN\Administrators	None	True	None	FullControl

Figure 25-9

Now let's look at actually modifying the ACL lists. In this example, you give a user explicit Full Control rights over the `TextFile.txt` file. You can use either an existing user or create a new test User account in Windows to run this sample. Listing 25-11 shows how to add an access rule to the `TextFile.txt` file.

Listing 25-11: Adding a rule to the Access Control List

VB

```
Dim sec As System.Security.AccessControl.FileSecurity = _
         System.IO.File.GetAccessControl(Server.MapPath("TextFile.txt"))

sec.AddAccessRule( _
         New System.Security.AccessControl.FileSystemAccessRule( _
                  "DEMOXP\TestUser", _
                  System.Security.AccessControl.FileSystemRights.FullControl, _
                  System.Security.AccessControl.AccessControlType.Allow _
                  ) _
         )

System.IO.File.SetAccessControl(Server.MapPath("TextFile.txt"),sec)
```

C#

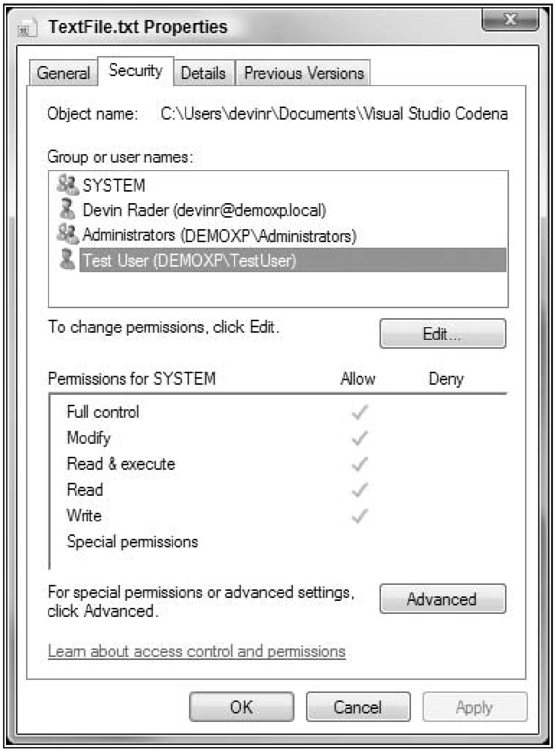
```
System.Security.AccessControl.FileSecurity sec =
         System.IO.File.GetAccessControl(Server.MapPath("TextFile.txt"));

sec.AddAccessRule(
         new System.Security.AccessControl.FileSystemAccessRule(
                  @"DEMOXP\TestUser",
                  System.Security.AccessControl.FileSystemRights.FullControl,
                  System.Security.AccessControl.AccessControlType.Allow
                  )
         );

System.IO.File.SetAccessControl(Server.MapPath("TextFile.txt"),sec);
```

There are several things to notice in this code sample. First, notice that you are passing three parameters to the `FileSystemAccessRule` constructor. The first parameter is the user you want to give rights to; change this value to a user on your specific system. Also notice that you must specify the full DOMAIN\USERNAME for the user. Next, notice that, in the code, you are using the `FileSystemRights` enumeration to specify exactly which rights you want to give to this user. You can specify multiple rights by using a bitwise `Or` operator, as shown in the following:

```
new System.Security.AccessControl.FileSystemAccessRule(
    "DEMOXP\TestUser",
    System.Security.AccessControl.FileSystemRights.Read &
        System.Security.AccessControl.FileSystemRights.Write,
    System.Security.AccessControl.AccessControlType.Allow
)
```

After running Listing 25-11, take a look at the Security tab in the file's Properties dialog and you should see that the user has been added to the Access Control List and allowed Full Control. Figure 25-10 shows what the dialog should look like.

Now remove the ACL you just added by running essentially the same code, but using the `RemoveAccessRule` method rather than the `AddAccessRule` method. Listing 25-12 shows this code.


TextFile.txt Properties
General
Security
Details
Previous Versions
Edit...
Advanced
Learn about access control and permissions
OK
Cancel
Apply

Object name: C:\Users\devinr\Documents\Visual Studio Codena

Group or user names:

- SYSTEM
- Devin Rader (devinr@demoxp.local)
- Administrators (DEMOXP\Administrators)
- Test User (DEMOXP\TestUser)

To change permissions, click Edit.

Permissions for SYSTEM	Allow	Deny
Full control	✓	
Modify	✓	
Read & execute	✓	
Read	✓	
Write	✓	
Special permissions		

For special permissions or advanced settings, click Advanced.

Figure 25-10

Listing 25-12: Removing the rule from the Access Control List

VB

```
Dim sec As System.Security.AccessControl.FileSecurity = _
        System.IO.File.GetAccessControl(Server.MapPath("TextFile.txt"))

sec.RemoveAccessRule( _
        new System.Security.AccessControl.FileSystemAccessRule( _
                "DEMOXP\TestUser", _
                System.Security.AccessControl.FileSystemRights.FullControl, _
                System.Security.AccessControl.AccessControlType.Allow _
                ) _
        )

System.IO.File.SetAccessControl(Server.MapPath("TextFile.txt"),sec)
```

C#

```
System.Security.AccessControl.FileSecurity sec =
        System.IO.File.GetAccessControl(Server.MapPath("TextFile.txt"));
```

Continued

```
sec.RemoveAccessRule(
        new System.Security.AccessControl.FileSystemAccessRule(
                @"DEMOXP\TestUser",
                System.Security.AccessControl.FileSystemRights.FullControl,
                System.Security.AccessControl.AccessControlType.Allow)
        );

System.IO.File.SetAccessControl(Server.MapPath("TextFile.txt"),sec);
```

If you open the file Properties dialog again, you see that the user has been removed from the Access Control List.

Reading and Writing Files

Now that you have learned how to manage the files on the local system, this section shows you how to use the .NET Framework to perform input/output (I/O) operations, such as reading and writing, on those files. The .NET Framework makes performing I/O very easy because it uses a common model of reading or writing I/O data; so regardless of the source, virtually the same code can be used. The model is based on two basic concepts, Stream classes and Reader/Writer classes. Figure 25-11 shows the basic I/O model .NET uses and how Streams, Readers, and Writers work together to make it possible to transfer data to and from any number of sources in any number of formats. Note that the diagram shows only some of the Streams and Reader/Writer pairs in the .NET Framework.

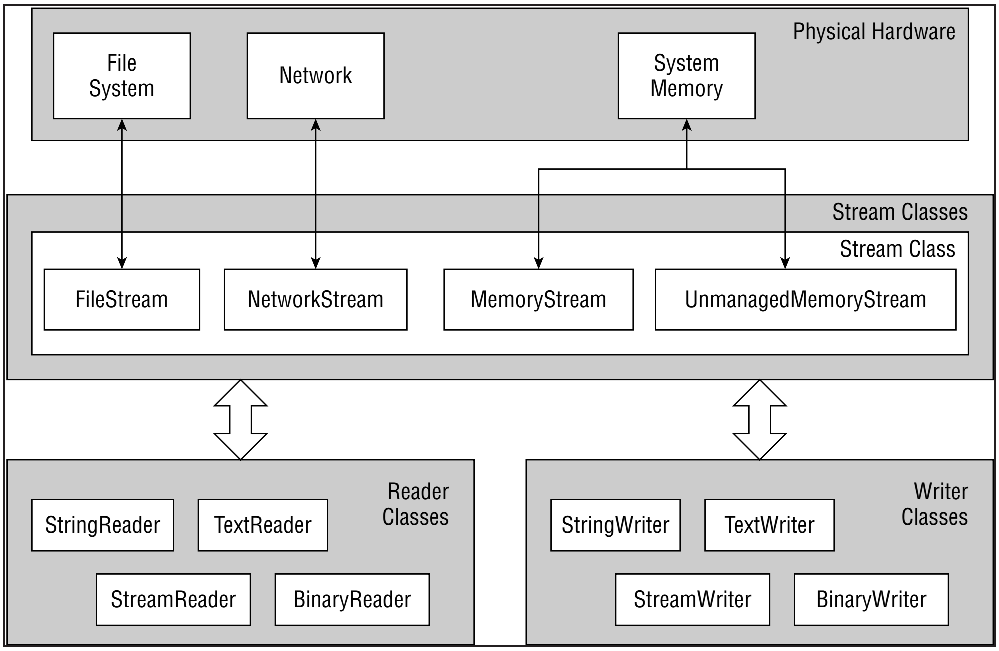

Physical Hardware
File System
Network
System Memory
Stream Classes
Stream Class
FileStream
NetworkStream
MemoryStream
UnmanagedMemoryStream
Reader Classes
StringReader
TextReader
StreamReader
BinaryReader
Writer Classes
StringWriter
TextWriter
StreamWriter
BinaryWriter

Figure 25-11

In this section, you dive deeper into learning how Streams, Readers, and Writers work and how .NET makes it easy to use them to transfer data.

Streams

Regardless of the type of I/O operation you are performing in .NET, if you want to read or write data you eventually use a stream of some type. Streams are the basic mechanism .NET uses to transfer data to and from its underlying source, be it a file, communication pipe, or TCP/IP socket. The `Stream` class provides the basic functionality to read and write I/O data, but because the `Stream` class is marked as abstract, you most likely need to use one of the several classes derived from `Stream`. Each `Stream` derivation is specialized to make it easy to transfer data from a specific source. The following table lists some of the classes derived from the `Stream` class.

Class	Description
`System.IO.FileStream`	Reads and writes files on a file system, as well as other file-related operating system handles (including pipes, standard input, standard output, and so on).
`System.IO.MemoryStream`	Creates streams that have memory as a backing store instead of a disk or a network connection. This can be useful in eliminating the need to write temporary files to disk or to store binary blob information in a database.
`System.IO.UnmanagedMemoryStream`	Supports access to unmanaged memory using the existing stream-based model and does not require that the contents in the unmanaged memory be copied to the heap.
`System.IO.BufferedStream`	Extends the `Stream` class by adding a buffering layer to read and write operations on another stream. The stream performs reads and writes in blocks (4096 bytes by default), which can result in improved efficiency.
`System.Net.Sockets.NetworkStream`	Implements the standard .NET Framework stream to send and receive data through network sockets. It supports both synchronous and asynchronous access to the network data stream.
`System.Security.Cryptography.CryptoStream`	Enables you to read and write data through cryptographic transformations.
`System.IO.Compression.GZipStream`	Enables you to compress data using the GZip data format.
`System.IO.Compression.DeflateStream`	Enables you to compress data using the `Deflate` algorithm. For more information, see the RFC 1951: DEFLATE 1.3 Specification.
`System.Net.Security.NegotiateStream`	Uses the Negotiate security protocol to authenticate the client, and optionally the server, in client-server communication.
`System.Net.Security.SslStream`	Necessary for client-server communication that uses the Secure Socket Layer (SSL) security protocol to authenticate the server and optionally the client.

As an example, you can use the `FileStream` to read a local system file from disk. To prepare for this sample, open the `TextFile.txt` you created for the samples in the previous section, enter some text, and save the file. Listing 25-13 shows the code to read this simple text file.

Listing 25-13: Using a FileStream to read a system file

VB

```
Dim fs As New FileStream (Server.MapPath("TextFile.txt"), FileMode.Open)
Dim data(fs.Length) As Byte
fs.Read(data, 0, fs.Length)
fs.Close()
```

C#

```
FileStream fs = new FileStream(Server.MapPath("TextFile.txt"), FileMode.Open);
byte[] data = new byte[fs.Length];
fs.Read(data, 0, (int)fs.Length);
fs.Close();
```

There are several items of note in this code. First, notice that you are creating a byte array the length of the stream, using the `Length` property to properly size the array, and then passing it to the `Read` method. The `Read` method fills the byte array with the stream data, in this case reading the entire stream into the byte array. If you want to read only a chunk of the stream or to start at a specific point in the stream, just change the parameters you pass to the `Read` method.

> *Streams use byte arrays as the basic means of transporting data to and from the underlying data source. You use a byte array to read data in this sample and, later in the chapter, you learn how to create a byte array that contains data you can write to a stream.*

Second, note that you are explicitly closing the `FileStream` using the `Close` method. Streams must always be explicitly closed in order to release the resources they are using, which in this case is the file. Failing to explicitly close the stream can cause memory leaks, and it may also deny other users and applications access to the resource.

> *A good way to ensure that your streams will always be closed once you are done using them is to wrap them in a* `Using` *statement.* `Using` *automatically calls the stream objects* `Dispose()` *method once the* `Using` *statement is closed. For the stream object, calling the* `Dispose` *method also automatically calls the streams* `Close()` *method. Utilizing the* `Using` *statement with stream objects is a good way to ensure that even if you do forget to explicitly add a call to close the stream, the object will be closed and the underlying resources released before the object is disposed.*

Finally, notice that in the `FileStream` constructor, you are passing two parameters, the first being the path to the file you want to read and the other indicating the type of access you want to use when opening the file. The `FileMode` enumeration lets you specify how the stream should be opened, for reading, writing, or both reading and writing.

Thinking about *how* you will use the opened file can become very important. Here are some issues you might want to consider when working with files using `FileStream`:

- ☐ Will you be reading, writing, or both?
- ☐ Are you creating a new file, or appending or truncating an existing file?
- ☐ Should other programs be allowed to access the file while you are using it?
- ☐ How are you going to read or write the data in the file? Are you looking for a specific location in the file, or simply reading the entire file from beginning to end?

Thankfully, the `FileStream` constructor includes a number of overloads that let you explicitly specify how you will use the file. The IO namespace also includes four enumerations that can help you control how the `FileStream` accesses your file:

- **FileMode:** The `FileMode` enumeration lets you control whether the file is appended, truncated, created, or opened.
- **FileAccess:** The `FileAccess` enumeration controls whether the file is opened for reading, writing, or both.
- **FileOptions:** The `FileOptions` enumeration controls several other miscellaneous options, such as random or sequential access, file encryption, or asynchronous file writing.
- **FileShare:** The `FileShare` enumeration controls the access that other users and programs have to the file while your application is using it.

Listing 25-14 shows how you can use all these enumerations in the `FileStream` constructor to write data to the text file you created earlier. Notice that you are supplying the `FileStream` constructor with much more information on how you want to open the file. In this sample, you append another text string to the file you just read. To do this, set the `FileMode` to Append and the `FileAccess` to Write.

Listing 25-14: Using I/O enumerations to control file behavior when writing a file

VB

```
Dim fs As New System.IO.FileStream(Server.MapPath("TextFile.txt"), _
    System.IO.FileMode.Append, System.IO.FileAccess.Write, _
    System.IO.FileShare.Read, 8, System.IO.FileOptions.None)
Dim data() As Byte =  _
               System.Text.Encoding.ASCII.GetBytes("This is an additional string")
fs.Write(data, 0, data.Length)
fs.Flush()
fs.Close()
```

C#

```
System.IO.FileStream fs =
    new System.IO.FileStream(Server.MapPath("TextFile.txt"),
        System.IO.FileMode.Append, System.IO.FileAccess.Write,
        System.IO.FileShare.Read, 8, System.IO.FileOptions.None);
byte[] data = System.Text.Encoding.ASCII.GetBytes("This is an additional string");
fs.Write(data, 0, data.Length);
fs.Flush();
fs.Close();
```

You can write your text to the file by encoding a string to a byte array, which contains the information you want to write. Then, using the `Write` method, write your byte array to the `FileStreams` buffer and use the `Flush` method to instruct the `FileStream` to clear its buffer, causing any buffered data to be committed to the underlying data store. Finally, close the `FileStream`, releasing any resources it is using. If you open the `TextFile.txt` file in Notepad, you should see your string has been appended to the existing text in the file.

Note that using the `Flush` method in this scenario is optional because the `Close` method also calls `Flush` internally to commit the data to the data store. However, because the `Flush` method does not release the

`FileStream` resources as `Close` does, it can be very useful if you are going to perform multiple write operations and do not want to release and then reacquire the resources for each write operation.

As you can see, so far reading and writing to files is really quite easy. The good thing is that, as mentioned earlier, because .NET uses the same basic `Stream` model for a variety of data stores, you can use these same techniques for reading and writing to any of the `Stream` derived classes. Listing 25-15 shows how you can use the same basic code to write to a `MemoryStream`, and Listing 25-16 demonstrates reading a Telnet server response using the `NetworkStream`.

Listing 25-15: Writing to a MemoryStream

VB

```
Dim data() As Byte = System.Text.Encoding.ASCII.GetBytes("This is a string")
Dim ms As New System.IO.MemoryStream()
ms.Write(data, 0, data.Length)
ms.Close()
```

C#

```
byte[] data = System.Text.Encoding.ASCII.GetBytes("This is a string");
System.IO.MemoryStream ms = new System.IO.MemoryStream();
ms.Write(data, 0, data.Length);
ms.Close();
```

Listing 25-16: Reading from a NetworkStream

VB

```
Dim client As New System.Net.Sockets.TcpClient()

' Note: You can find a large list of Telnet accessible
' BBS systems at http://www.dmine.com/telnet/brieflist.htm

' The WCS Online BBS (http://bbs.wcssoft.com)
Dim addr As System.Net.IPAddress = System.Net.IPAddress.Parse("65.182.234.52")
Dim endpoint As New System.Net.IPEndPoint(addr, 23)

client.Connect(endpoint)
Dim ns As System.Net.Sockets.NetworkStream = client.GetStream()

If (ns.DataAvailable) Then
     Dim data(client.ReceiveBufferSize) As Byte
     ns.Read(data, 0, client.ReceiveBufferSize)
     Dim response As String = System.Text.Encoding.ASCII.GetString(data)
End If
ns.Close()
```

C#

```
System.Net.Sockets.TcpClient client = new System.Net.Sockets.TcpClient();

// Note: You can find a large list of Telnet accessible
// BBS systems at http://www.dmine.com/telnet/brieflist.htm

// The WCS Online BBS (http://bbs.wcssoft.com)
System.Net.IPAddress addr = System.Net.IPAddress.Parse("65.182.234.52");
System.Net.IPEndPoint endpoint = new System.Net.IPEndPoint(addr,23);
```

```
client.Connect(endpoint);
System.Net.Sockets.NetworkStream ns = client.GetStream();

if (ns.DataAvailable)
{
    byte[] bytes = new byte[client.ReceiveBufferSize];
    ns.Read(bytes, 0, client.ReceiveBufferSize);
    string data = System.Text.Encoding.ASCII.GetString(bytes);
}
ns.Close();
```

Notice that the concept in both examples is virtually identical. You create a `Stream` object, read the bytes into a byte array for processing, and then close the stream. The code varies only in the implementation of specific Streams.

Readers and Writers

Other main parts of I/O in the .NET Framework are `Reader` and `Writer` classes. These classes help insulate you from having to deal with reading and writing individual bytes to and from Streams, enabling you to concentrate on the data you are working with. The .NET Framework provides a wide variety of reader and writer classes, each designed for reading or writing according to a specific set of rules. The first table following shows a partial list of the readers available in the .NET Framework. The second table lists the corresponding writer classes.

Class	Description
`System.IO.TextReader`	Abstract class that enables the reading of a sequential series of characters.
`System.IO.StreamReader`	Reads characters from a byte stream. Derived from `TextReader`.
`System.IO.StringReader`	Reads textual information as a stream of in-memory characters. Derived from `TextReader`.
`System.IO.BinaryReader`	Reads primitive data types as binary values from a stream.
`System.Xml.XmlTextReader`	Provides fast, non-cached, forward-only access to XML.

Class	Description
`System.IO.TextWriter`	Abstract class that enables the writing of a sequential series of characters.
`System.IO.StreamWriter`	Writes characters to a stream. Derived from `TextWriter`.
`System.IO.StringWriter`	Writes textual information as a stream of in-memory characters. Derived from `TextWriter`.

Class	Description
`System.IO .BinaryWriter`	Writes primitive data types in binary to a stream.
`System.Xml .XmlTextWriter`	Provides a fast, non-cached, forward-only way of generating XML streams or files.

Now look at using several different types of readers and writers, starting with a simple example. Listing 25-17 shows you how to use a `StreamReader` to read a `FileStream`.

Listing 25-17: Reading and writing a text file with a StreamReader

VB

```
Dim streamwriter As New System.IO.StreamWriter( _
    System.IO.File.Open("C:\Wrox\temp.txt", System.IO.FileMode.Open) )
streamwriter.Write("This is a string")
streamwriter.Close()

Dim reader As New System.IO.StreamReader( _
    System.IO.File.Open("C:\Wrox\temp.txt", System.IO.FileMode.Open) )
Dim tmp As String = reader.ReadToEnd()
reader.Close()
```

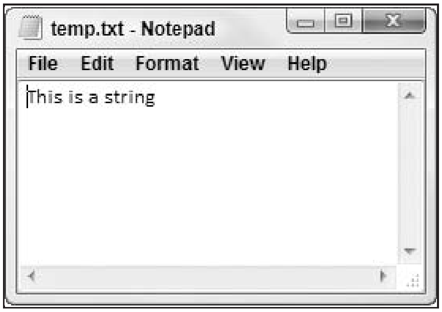
C#

```
System.IO.StreamWriter streamwriter =
             new System.IO.StreamWriter(
             System.IO.File.Open(@"C:\Wrox\temp.txt", System.IO.FileMode.Open) );
streamwriter.Write("This is a string");
streamwriter.Close();

System.IO.StreamReader reader =
             new System.IO.StreamReader(
             System.IO.File.Open(@"C:\Wrox\temp.txt",System.IO.FileMode.Open) );
string tmp = reader.ReadToEnd();
reader.Close();
```

Notice that when you create a `StreamReader`, you must pass an existing stream instance as a constructor parameter. The reader uses this stream as its underlying data source. In this sample, you use the `File` class's static `Open` method to open a writable `FileStream` for your `StreamWriter`.

Also notice that you no longer have to deal with byte arrays. The `StreamReader` takes care of converting the data to a type that's more user-friendly than a byte array. In this case, you are using the `ReadToEnd` method to read the entire stream and convert it to a string. The `StreamReader` provides a number of different methods for reading data that you can use depending on exactly how you want to read the data, from reading a single character using the `Read` method, to reading the entire file using the `ReadToEnd` method.

Figure 25-12 shows the results of your write when you open the file in Notepad.



This is a string

Figure 25-12

Now use the `BinaryReader` and `BinaryWriter` classes to read and write some primitive types to a file. The `BinaryWriter` writes primitive objects in their native format, so in order to read them using the `BinaryReader`, you must select the appropriate `Read` method. Listing 25-18 shows you how to do that; in this case, you are writing a value from a number of different primitive types to the text file, then reading the same value.

Listing 25-18: Reading and writing binary data

VB

```
Dim binarywriter As New System.IO.BinaryWriter( _
    System.IO.File.Create("C:\Wrox\binary.dat"))

binarywriter.Write("a string")
binarywriter.Write(&H12346789ABCDEF)
binarywriter.Write(&H12345678)
binarywriter.Write("c"c)
binarywriter.Write(1.5F)
binarywriter.Write(100.2D)
binarywriter.Close()

Dim binaryreader As New System.IO.BinaryReader( _
    System.IO.File.Open("C:\Wrox\binary.dat", System.IO.FileMode.Open))

Dim a As String = binaryreader.ReadString()
Dim l As Long = binaryreader.ReadInt64()
Dim i As Integer = binaryreader.ReadInt32()
Dim c As Char = binaryreader.ReadChar()
Dim f As Double = binaryreader.ReadSingle()
Dim d As Decimal = binaryreader.ReadDecimal()
binaryreader.Close()
```

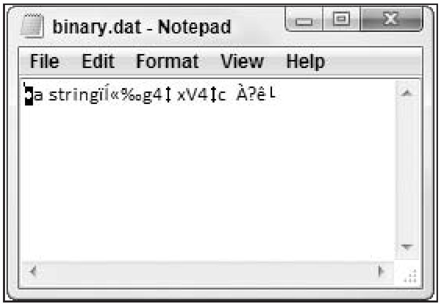
C#

```
System.IO.BinaryWriter binarywriter =
            new System.IO.BinaryWriter(
                System.IO.File.Create(@"C:\Wrox\binary.dat") );
binarywriter.Write("a string");
binarywriter.Write(0x12346789abcdef);
```

Continued

```
binarywriter.Write(0x12345678);
binarywriter.Write('c');
binarywriter.Write(1.5f);
binarywriter.Write(100.2m);
binarywriter.Close();

System.IO.BinaryReader binaryreader =
            new System.IO.BinaryReader(
                System.IO.File.Open(@"C:\Wrox\binary.dat",
                    System.IO.FileMode.Open));

string a = binaryreader.ReadString();
long l = binaryreader.ReadInt64();
int i = binaryreader.ReadInt32();
char c = binaryreader.ReadChar();
float f = binaryreader.ReadSingle();
decimal d = binaryreader.ReadDecimal();
binaryreader.Close();
```

If you open this file in Notepad, you should see that the `BinaryWriter` has written the nonreadable binary data to the file. Figure 25-13 shows what the content of the file looks like. The `BinaryReader` provides a number of different methods for reading various kinds of primitive types from the stream.
In this sample, you use a different `Read` method for each primitive type that you write to the file.



```
a stringïÍ«‰g4↕ xV4↕c À?ê└
```

Figure 25-13

Finally, notice that the basic usage of both the `StreamReader`/`StreamWriter` and `BinaryReader`/`BinaryWriter` classes is virtually identical. You can apply the same basic ideas to use any of the reader or writer classes.

Encodings

The `StreamReader` by default attempts to determine the encoding format of the file. If one of the supported encodings such as UTF-8 or UNICODE is detected, it is used. If the encoding is not recognized, the default encoding of UTF-8 is used. Depending on the constructor you call, you can change the default encoding used and optionally turn off encoding detection. The following example shows how you can control the encoding that the `StreamReader` uses.

```
StreamReader reader =
           new StreamReader(@"C:\Wrox\text.txt",System.Text.Encoding.Unicode);
```

The default encoding for the `StreamWriter` is also UTF-8, and you can override it in the same manner as the `StreamReader` class.

I/O Shortcuts

Although knowing how to create and use streams is always very useful and worth studying, the .NET Framework provides you with numerous shortcuts for common tasks like reading and writing to files. For instance, if you want to read the entire file, you can simply use one of the static Read All methods of the `File` class. Using these methods, you cause .NET to handle the process of creating the `Stream` and `StreamReader` for you, and simply return the resulting string of data. This is just one example of the shortcuts that the .NET Framework provides. Listing 25-19 shows some of the others, with explanatory comments. Keep in mind that Listing 25-19 is showing individual code snippets; do not try to run the listing as a single block of code.

Listing 25-19: Using the static method of the File and Directory classes

VB

```
' Opens a file and returns a FileStream
Dim fs As System.IO FileStream = _
    System.IO.File.Open("C:\Wrox\temp.txt", System.IO.FileMode.Open)


' Opens a file and returns a StreamReader for reading the data
Dim sr As System.IO.StreamReader = System.IO.File.OpenText("C:\Wrox\temp.txt")


' Opens a filestream for reading
Dim fs As System.IO.FileStream = System.IO.File.OpenRead("C:\Wrox\temp.txt")


' Opens a filestream for writing
Dim fs As System.IO.FileStream = System.IO.File.OpenWrite("C:\Wrox\temp.txt")


' Reads the entire file and returns a string of data
Dim data As String = System.IO.File.ReadAllText("C:\Wrox\temp.txt")


' Writes the string of data to the file
System.IO.File.WriteAllText("C:\Wrox\temp.txt", data)
```

C#

```
// Opens a file and returns a FileStream
System.IO.FileStream fs =
    System.IO.File.Open(@"C:\Wrox\temp.txt", System.IO.FileMode.Open);


// Opens a file and returns a StreamReader for reading the data
System.IO.StreamReader sr = System.IO.File.OpenText(@"C:\Wrox\temp.txt");


// Opens a filestream for reading
```

Continued

```
System.IO.FileStream fs = System.IO.File.OpenRead(@"C:\Wrox\temp.txt");


// Opens a filestream for writing
System.IO.FileStream fs = System.IO.File.OpenWrite(@"C:\Wrox\temp.txt");


// Reads the entire file and returns a string of data
string data = System.IO.File.ReadAllText(@"C:\Wrox\temp.txt");


// Writes the string of data to the file
System.IO.File.WriteAllText(@"C:\Wrox\temp.txt", data);
```

Compressing Streams

Introduced in the .NET 2.0 Framework, the System.IO.Compression namespace includes classes for compressing and decompressing data using either the `GZipStream` or the `DeflateStream` classes.

GZip Compression

Because both new classes are derived from the `Stream` class, using them should be relatively similar to using the other `Stream` operations you have examined so far in this chapter. Listing 25-20 shows an example of compressing your text file using the `GZipStream` class.

Listing 25-20: Compressing a file using GZipStream

VB

```
' Read the file we are going to compress into a FileStream
Dim filename As String = Server.MapPath("TextFile.txt")

Dim infile As System.IO.FileStream = System.IO.File.OpenRead(filename)
Dim buffer(infile.Length) As Byte
infile.Read(buffer, 0, buffer.Length)
infile.Close()

' Create the output file
Dim outfile As System.IO.FileStream = _
    System.IO.File.Create(System.IO.Path.ChangeExtension(filename, "zip"))

' Compress the input stream and write it to the output FileStream
Dim gzipStream As New System.IO.Compression.GZipStream(
    outfile, System.IO.Compression.CompressionMode.Compress)
gzipStream.Write(buffer, 0, buffer.Length)
gzipStream.Close()
```

C#

```
// Read the file we are going to compress into a FileStream
string filename = Server.MapPath("TextFile.txt");

System.IO.FileStream infile = System.IO.File.OpenRead(filename);
byte[] buffer = new byte[infile.Length];
```

```
infile.Read(buffer, 0, buffer.Length);
infile.Close();

// Create the output file
System.IO.FileStream outfile =
    System.IO.File.Create(System.IO.Path.ChangeExtension(filename, "zip"));

// Compress the input stream and write it to the output FileStream
System.IO.Compression.GZipStream gzipStream =
    new System.IO.Compression.GZipStream(outfile,
        System.IO.Compression.CompressionMode.Compress);
gzipStream.Write(buffer, 0, buffer.Length);
gzipStream.Close();
```

Notice that the `GZipStream` constructor requires two parameters, the stream to write the compressed data to, and the `CompressionMode` enumeration, which tells the class if you want to compress or decompress data. After the code runs, be sure there is a file called `text.zip` in your Web site directory.

Deflate Compression

The `Compression` namespace also allows to you decompress a file using the `GZip` or `Deflate` methods. Listing 25-21 shows an example of decompressing a file using the `Deflate` method.

Listing 25-21: Decompressing a file using DeflateStream

VB

```
Dim filename As String = Server.MapPath("TextFile.zip")

Dim infile As System.IO.FileStream = System.IO.File.OpenRead(filename)
Dim deflateStream As New System.IO.Compression.DeflateStream( _
    infile, System.IO.Compression.CompressionMode.Decompress)
Dim buffer(infile.Length + 100) As Byte

Dim offset As Integer = 0
Dim totalCount As Integer = 0
While True
    Dim bytesRead As Integer = deflateStream.Read(buffer, offset, 100)
    If bytesRead = 0 Then
        Exit While
    End If
    offset += bytesRead
    totalCount += bytesRead
End While

Dim outfile As System.IO.FileStream = _
    System.IO.File.Create(System.IO.Path.ChangeExtension(filename, "txt"))
outfile.Write(buffer, 0, buffer.Length)
outfile.Close()
```

C#

```
string filename = Server.MapPath("TextFile.zip");

System.IO.FileStream infile = System.IO.File.OpenRead(filename);
```

Continued

```
System.IO.Compression.DeflateStream deflateStream =
    new System.IO.Compression.DeflateStream(infile,
        System.IO.Compression.CompressionMode.Decompress);
byte[] buffer = new byte[infile.Length + 100];

int offset = 0;
int totalCount = 0;
while (true)
{
    int bytesRead = deflateStream.Read(buffer, offset, 100);
    if (bytesRead == 0)
    { break; }

    offset += bytesRead;
    totalCount += bytesRead;
}

System.IO.FileStream outfile =
    System.IO.File.Create(System.IO.Path.ChangeExtension(filename, "txt"));
outfile.Write(buffer, 0, buffer.Length);
outfile.Close();
```

Compressing HTTP Output

Besides compressing files, one other very good use of the compression features of the .NET Framework in an ASP.NET application is to implement your own `HttpModule` class that compresses the HTTP output of your application. This is easier than it might sound, and it will save you precious bandwidth by compressing the data that is sent from your Web server to the browsers that support the HTTP 1.1 Protocol standard (which most do). The browser can then decompress the data before rendering it.

> *IIS 6 does offer built-in HTTP compression capabilities, and there are several third-party HTTP compression modules available, such as the Blowery Http Compression Module (`www.blowery.org`).*

Start by creating a Windows Class library project. Add a new class to your project called `CompressionModule`. This class is your compression `HttpModule`. Listing 25-22 shows the code for creating the class.

Listing 25-22: Compressing HTTP output with an HttpModule

VB

```
Imports System
Imports System.Collections.Generic
Imports System.Text
Imports System.Web
Imports System.IO
Imports System.IO.Compression

Namespace Wrox.Demo.Compression

Public Class CompressionModule
    Implements IHttpModule
```

Continued

```
    Public Sub Dispose() Implements System.Web.IHttpModule.Dispose
        Throw New Exception("The method or operation is not implemented.")
    End Sub

    Public Sub Init(ByVal context As System.Web.HttpApplication) _
       Implements System.Web.IHttpModule.Init
        AddHandler context.BeginRequest, AddressOf context_BeginRequest
    End Sub

    Public Sub context_BeginRequest(ByVal sender As Object, ByVal e As EventArgs)
        Dim app As HttpApplication = CType(sender, HttpApplication)

        'Get the Accept-Encoding HTTP header from the request.
        'The requesting browser sends this header which we will use
        ' to determine if it supports compression, and if so, what type
        ' of compression algorithm it supports
        Dim encodings As String = app.Request.Headers.Get("Accept-Encoding")

        If (encodings = Nothing) Then
            Return
        End If

        Dim s As Stream = app.Response.Filter

        encodings = encodings.ToLower()

        If (encodings.Contains("gzip")) Then
            app.Response.Filter = New GZipStream(s, CompressionMode.Compress)
            app.Response.AppendHeader("Content-Encoding", "gzip")
            app.Context.Trace.Warn("GZIP Compression on")
        Else
            app.Response.Filter = _
                          New DeflateStream(s, CompressionMode.Compress)
            app.Response.AppendHeader("Content-Encoding", "deflate")
            app.Context.Trace.Warn("Deflate Compression on")
        End If
    End Sub
End Class

End Namespace
```

C#

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web;
using System.IO;
using System.IO.Compression;

namespace Wrox.Demo.Compression
{
    public class CompressionModule : IHttpModule
```

Continued

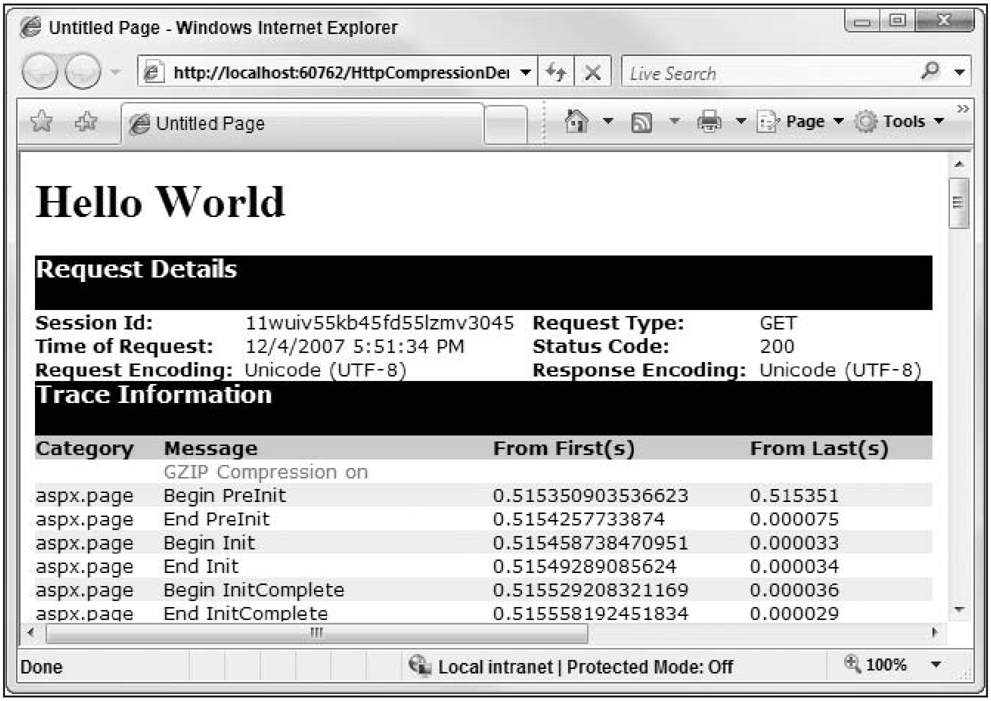
```
    {

        #region IHttpModule Members

        void IHttpModule.Dispose()
        {
            throw new Exception("The method or operation is not implemented.");
        }

        void IHttpModule.Init(HttpApplication context)
        {
            context.BeginRequest += new EventHandler(context_BeginRequest);
        }

        void context_BeginRequest(object sender, EventArgs e)
        {
            HttpApplication app = (HttpApplication)sender;

            //Get the Accept-Encoding HTTP header from the request.
            //The requesting browser sends this header which we will use
            // to determine if it supports compression, and if so, what type
            // of compression algorithm it supports
            string encodings = app.Request.Headers.Get("Accept-Encoding");

            if (encodings == null)
                return;

            Stream s = app.Response.Filter;

            encodings = encodings.ToLower();

            if (encodings.Contains("gzip"))
            {
                app.Response.Filter = new GZipStream(s, CompressionMode.Compress);
                app.Response.AppendHeader("Content-Encoding", "gzip");
                app.Context.Trace.Warn("GZIP Compression on");
            }
            else
            {
                app.Response.Filter =
                              new DeflateStream(s, CompressionMode.Compress);
                app.Response.AppendHeader("Content-Encoding", "deflate");
                app.Context.Trace.Warn("Deflate Compression on");
            }
        }

        #endregion
    }
}
```

After you create and build the module, add the assembly to your Web site's `Bin` directory. After that's done, you let your Web application know that it should use the `HttpModule` when it runs. Do this by adding the module to the `web.config` file. Listing 25-23 shows the nodes to add to the `web.config` `system.web` configuration section.

Listing 25-23: Adding an HttpCompression module to the web.config

```
<httpModules>
      <add name="HttpCompressionModule"
       type="Wrox.Demo.Compression.CompressionModule, HttpCompressionModule"/>
</httpModules>

<trace enabled="true" />
```

Notice that one other change you are making is to enable page tracing. You use this to demonstrate that the page is actually being compressed. When you run the page, you should see the trace output shown in Figure 25-14. Notice a new entry under the trace information showing that the GZip compression has been enabled on this page.


Untitled Page - Windows Internet Explorer
http://localhost:60762/HttpCompressionDe
Live Search
Untitled Page
Page
Tools
Done
Local intranet | Protected Mode: Off
100%

Hello World

Request Details

Session Id:	11wuiv55kb45fd55lzmv3045	**Request Type:**	GET
Time of Request:	12/4/2007 5:51:34 PM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
	GZIP Compression on		
aspx.page	Begin PreInit	0.515350903536623	0.515351
aspx.page	End PreInit	0.5154257733874	0.000075
aspx.page	Begin Init	0.515458738470951	0.000033
aspx.page	End Init	0.51549289085624	0.000034
aspx.page	Begin InitComplete	0.515529208321169	0.000036
aspx.page	End InitComplete	0.515558192451834	0.000029

Figure 25-14

Working with Serial Ports

Also introduced in the .NET 2.0 Framework was the `System.IO.Ports` namespace. This namespace contains classes that enable you to work with and communicate through serial ports.

.NET provides a `SerialPort` component that you can add to the Component Designer of your Web page. Adding this component enables your application to communicate via the serial port. Listing 25-24 shows how to write some text to the serial port.

Listing 25-24: Writing text to the serial port

VB

```
<script runat="server">
    Dim SerialPort1 As New System.IO.Ports.SerialPort()

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)

        Me.SerialPort1.PortName = "COM1"

        If (Not Me.SerialPort1.IsOpen()) Then
            Me.SerialPort1.Open()
        End If

        Me.SerialPort1.Write("Hello World")
        Me.SerialPort1.Close()

    End Sub
</script>
```

C#

```
<script runat="server">

    System.IO.Ports.SerialPort SerialPort1 = new System.IO.Ports.SerialPort();

    protected void Page_Load(object sender, EventArgs e)
    {
        this.SerialPort1.PortName = "COM1";

        if (!this.SerialPort1.IsOpen)
        {
            this.SerialPort1.Open();
        }

        this.SerialPort1.Write("Hello World");
        this.SerialPort1.Close();
    }

</script>
```

This code simply attempts to open the serial port COM1 and write a bit of text. The `SerialPort` component gives you control over most aspects of the serial port, including baud rate, parity, and stop bits.

Network Communications

Finally, this chapter takes you beyond your own systems and talks about how you can use the .NET Framework to communicate with other systems. The .NET Framework contains a rich set of classes in the `System.Net` namespace that allow you to communicate over a network using a variety of protocols and communications layers. You can perform all types of actions, from DNS resolution to programmatic HTTP Posts to sending e-mail through SMTP.

WebRequest and WebResponse

The first series of classes to discuss are the `WebRequest` and `WebResponse` classes. You can use these two classes to develop applications that can make a request to a Uniform Resource Identifier (URI) and receive a response from that resource. The .NET Framework provides three derivatives of the `WebRequest` and `WebResponse` classes, each designed to communicate to a specific type of end point via HTTP, FTP, and file:// protocols.

HttpWebRequest and HttpWebResponse

The first pair of classes are the `HttpWebRequest` and `HttpWebResponse` classes. As you can probably guess based on their names, these two classes are designed to communicate using the HTTP protocol. Perhaps the most famous use of the `HttpWebRequest` and `HttpWebResponse` classes is to write applications that can make requests to other Web pages via HTTP and parse the resulting text to extract data. This is known as *screen scraping*.

For an example of using the `HttpWebRequest` and `HttpWebResponse` classes to screen scrape, you can use the following code to build a Web page that will serve as a simple Web browser. You also learn how another Web page can be displayed inside of yours using an `HttpWebRequest`. In this example, you scrape the wrox.com home page and display it in a panel on your Web page. Listing 25-25 shows the code.

Listing 25-25: Using an HttpWebRequest to retrieve a Web page

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace=System.IO %>
<%@ Import Namespace=System.Net %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim uri As New Uri("http://www.wrox.com/")
        If (uri.Scheme = uri.UriSchemeHttp) Then
            Dim request As HttpWebRequest = HttpWebRequest.Create(uri)
            request.Method = WebRequestMethods.Http.Get
            Dim response As HttpWebResponse = request.GetResponse()
            Dim reader As New StreamReader(response.GetResponseStream())
            Dim tmp As String = reader.ReadToEnd()
            response.Close()

            Me.Panel1.GroupingText = tmp
        End If
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
```

Continued

```
            <p>This is the wrox.com website:</p>
            <asp:Panel ID="Panel1" runat="server"
                Height="355px" Width="480px" ScrollBars=Auto>
            </asp:Panel>
        </div>
        </form>
    </body>
    </html>
```

C#

```
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Uri uri = new Uri("http://www.wrox.com/");
        if (uri.Scheme == Uri.UriSchemeHttp)
        {
            HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create( uri );
            request.Method = WebRequestMethods.Http.Get;
            HttpWebResponse response = (HttpWebResponse)request.GetResponse();
            StreamReader reader = new StreamReader(response.GetResponseStream());
            string tmp = reader.ReadToEnd();
            response.Close();

            this.Panel1.GroupingText = tmp;
        }
    }
</script>
```

Figure 25-15 shows what the Web page look likes when you execute the code in Listing 25-25. The `HttpWebRequest` to the Wrox.com home page returns a string containing the scraped HTML. The sample assigns the value of this string to the `GroupingText` property of the Panel control. When the final page is rendered, the browser renders the HTML that was scraped as literal content on the page.

One other use of the `HttpWebRequest` and `HttpWebResponse` classes is to programmatically post data to another Web page, as shown in Listing 25-26.

Listing 25-26: Using an HttpWebRequest to post data to a remote Web page

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace=System.IO %>
<%@ Import Namespace=System.Net %>

<script runat="server">

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim uri As New Uri("http://www.amazon.com/" & _
                        "exec/obidos/search-handle-form/102-5194535-6807312")
        Dim data As String = "field-keywords=Professional ASP.NET 3.5"
        If (uri.Scheme = uri.UriSchemeHttp) Then

            Dim request As HttpWebRequest = HttpWebRequest.Create(uri)
```

Continued

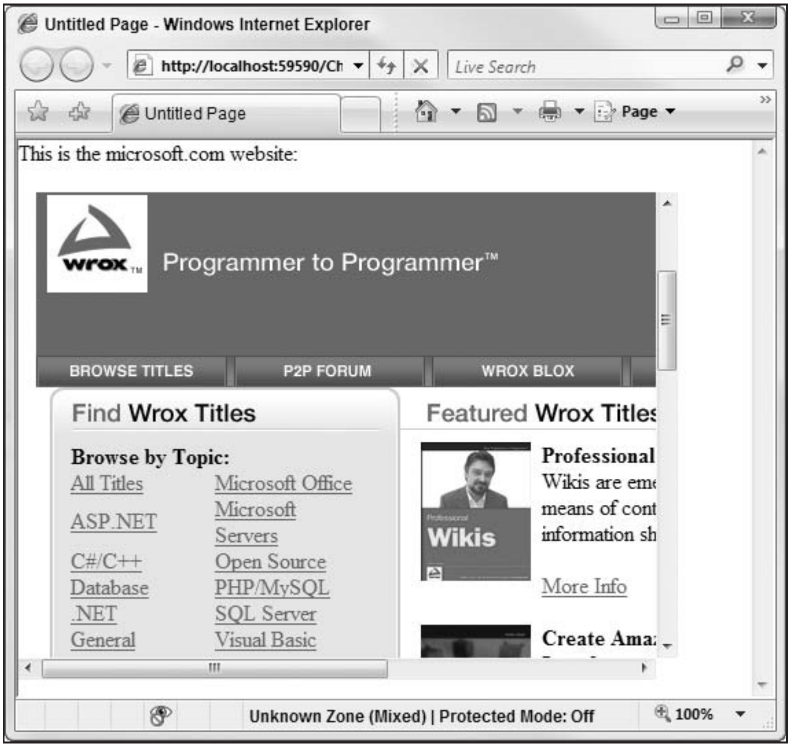
```
            request.Method = WebRequestMethods.Http.Post
            request.ContentLength = data.Length
            request.ContentType = "application/x-www-form-urlencoded"

            Dim writer As New StreamWriter(request.GetRequestStream())
            writer.Write(data)
            writer.Close()

            Dim response As HttpWebResponse = request.GetResponse()
            Dim reader As New StreamReader(response.GetResponseStream())
            Dim tmp As String = reader.ReadToEnd()
            response.Close()

            Me.Panel1.GroupingText = tmp
        End If
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Panel ID="Panel1" runat="server"
            Height="355px" Width="480px" ScrollBars=Auto>
        </asp:Panel>
    </div>
    </form>
</body>
</html>
```

C#

```
<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        Uri uri = new Uri("http://www.amazon.com/" +
                        "exec/obidos/search-handle-form/102-5194535-6807312");
        string data = "field-keywords=Professional ASP.NET 3.5";
        if (uri.Scheme == Uri.UriSchemeHttp)
        {
            HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create(uri);
            request.Method = WebRequestMethods.Http.Post;
            request.ContentLength = data.Length;
            request.ContentType = "application/x-www-form-urlencoded";

            StreamWriter writer = new StreamWriter( request.GetRequestStream() );
            writer.Write(data);
            writer.Close();

            HttpWebResponse response = (HttpWebResponse)request.GetResponse();
```

Continued

```
            StreamReader reader = new StreamReader(response.GetResponseStream());
            string tmp = reader.ReadToEnd();
            response.Close();

            this.Panel1.GroupingText = tmp;
        }
    }
</script>
```


This is the microsoft.com website:

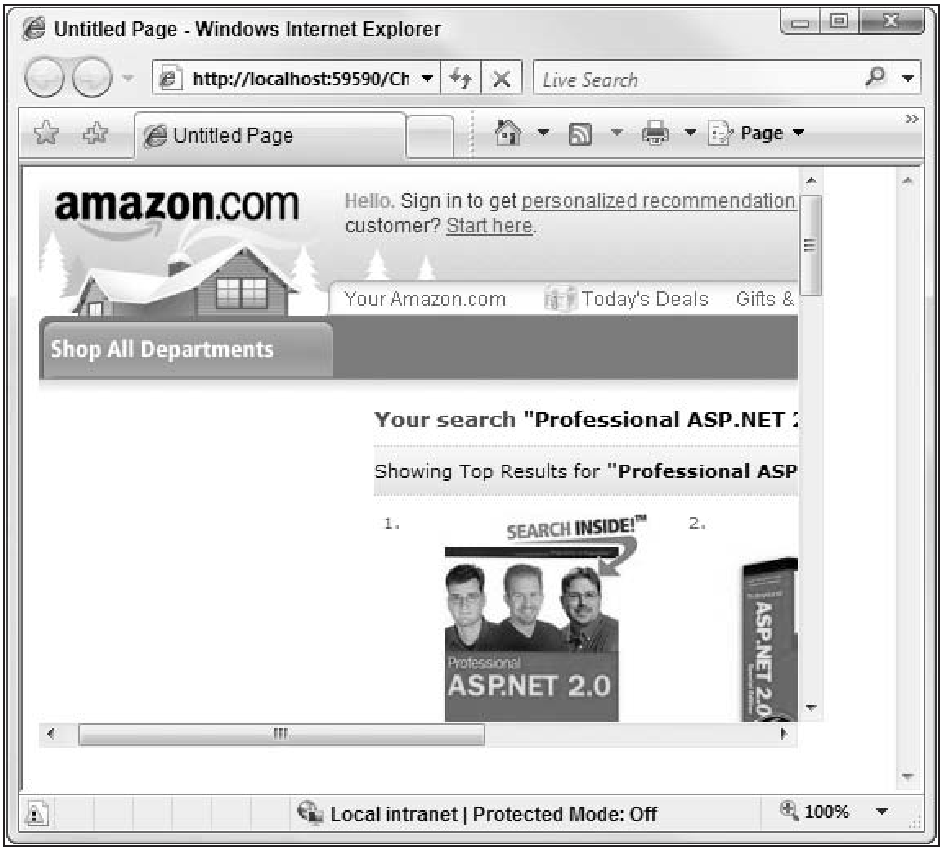
Untitled Page - Windows Internet Explorer
http://localhost:59590/Ch
Live Search
Untitled Page
Page
Programmer to Programmer™
BROWSE TITLES
P2P FORUM
WROX BLOX
Find Wrox Titles
Featured Wrox Titles
Browse by Topic:
All Titles
Microsoft Office
ASP.NET
Microsoft Servers
C#/C++
Open Source
Database
PHP/MySQL
.NET
SQL Server
General
Visual Basic
Professional
Wikis are eme
means of cont
information sh
More Info
Wikis
Create Ama
Unknown Zone (Mixed) | Protected Mode: Off
100%

Figure 25-15

You can see that the preceding code posts a search query to Amazon.com and receives the HTML as the response. As in the example shown earlier in Listing 25-25, you can simply use a Panel to display the resulting text as HTML. The results of the query are shown in Figure 25-16.

FtpWebRequest and FtpWebResponse

The next pair of classes are the `FtpWebRequest` and `FtpWebResponse` classes. These two classes were new additions to the .NET 2.0 Framework, and they make it easy to execute File Transfer Protocol (FTP) commands from your Web page. Using these classes, it is now possible to implement an entire FTP client right from your Web application. Listing 25-27 shows an example of downloading a text file from the public Microsoft.com FTP site. (See Figure 25-17.)

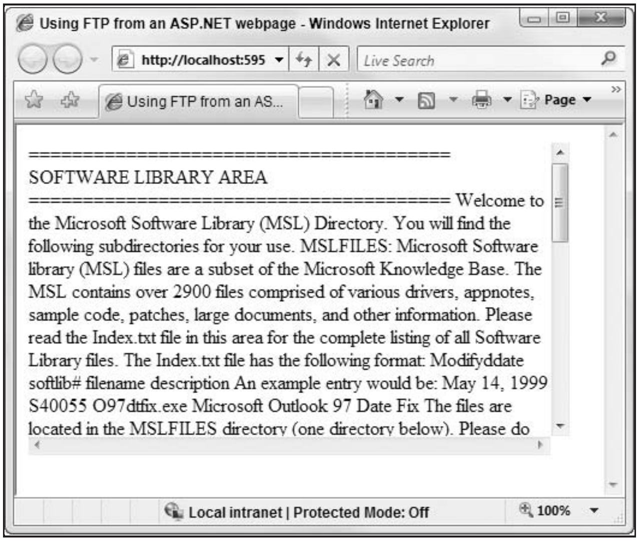

Untitled Page - Windows Internet Explorer
http://localhost:59590/Ch
Live Search
Untitled Page
Page
amazon.com
Your Amazon.com
Today's Deals
Gifts &
Shop All Departments
1.
2.
SEARCH INSIDE!™
Professional
ASP.NET 2.0
Local intranet | Protected Mode: Off
100%

Hello. Sign in to get personalized recommendation
customer? Start here.

Your search "Professional ASP.NET

Showing Top Results for "Professional ASP

Figure 25-16


Using FTP from an ASP.NET webpage - Windows Internet Explorer
http://localhost:595
Live Search
Using FTP from an AS...
Page
Local intranet | Protected Mode: Off
100%

== SOFTWARE LIBRARY AREA == Welcome to the Microsoft Software Library (MSL) Directory. You will find the following subdirectories for your use. MSLFILES: Microsoft Software library (MSL) files are a subset of the Microsoft Knowledge Base. The MSL contains over 2900 files comprised of various drivers, appnotes, sample code, patches, large documents, and other information. Please read the Index.txt file in this area for the complete listing of all Software Library files. The Index.txt file has the following format: Modifyddate softlib# filename description An example entry would be: May 14, 1999 S40055 O97dtfix.exe Microsoft Outlook 97 Date Fix The files are located in the MSLFILES directory (one directory below). Please do

Figure 25-17

Listing 25-27: Using an FtpWebRequest to download a file from an FTP site

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace=System.IO %>
<%@ Import Namespace=System.Net %>

<script runat="server">

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim uri As New Uri("ftp://ftp.microsoft.com/SoftLib/ReadMe.txt")
        If (uri.Scheme = uri.UriSchemeFtp) Then
            Dim request As FtpWebRequest = FtpWebRequest.Create(uri)
            request.Method = WebRequestMethods.Ftp.DownloadFile
            Dim response As FtpWebResponse = request.GetResponse()
            Dim reader As New StreamReader(response.GetResponseStream())
            Dim tmp As String = reader.ReadToEnd()
            response.Close()
            Me.Panel1.GroupingText = tmp
        End If
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Using FTP from an ASP.NET webpage</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <div runat="server" id="ftpContent"
            style="overflow:scroll; height: 260px; width: 450px;">
        </div>
    </div>
    </form>
</body>
</html>
```

C#

```
<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        Uri uri = new Uri("ftp://ftp.microsoft.com/SoftLib/ReadMe.txt ");
        if (uri.Scheme == Uri.UriSchemeFtp)
        {
            FtpWebRequest request = (FtpWebRequest)FtpWebRequest.Create(uri);
            request.Method = WebRequestMethods.Ftp.DownloadFile;
            FtpWebResponse response = (FtpWebResponse)request.GetResponse();
            StreamReader reader = new StreamReader(response.GetResponseStream());
```

Continued

```
            string tmp = reader.ReadToEnd();
            response.Close();

            this.Panel1.GroupingText = tmp;
        }
    }
</script>
```

FileWebRequest and FileWebResponse

Next, look at the `FileWebRequest` and `FileWebResponse` classes. These classes provide a file system implementation of the WebRequest and WebResponse classes and are designed to make it easy to transfer files using the file:// protocol, as shown in Listing 25-28.

Listing 25-28: Using the FileWebRequest to write to a remote file

VB

```
Dim uri As New Uri("file://DEMOXP/Documents/lorum.txt")
If (uri.Scheme = uri.UriSchemeFile) Then
    Dim request As System.Net.FileWebRequest = _
        System.Net.FileWebRequest.Create(uri)
    Dim response As System.Net.FileWebResponse = request.GetResponse()
    Dim reader As New System.IO.StreamReader(response.GetResponseStream())
    Dim tmp As String = reader.ReadToEnd()
    response.Close()
End If
```

C#

```
Uri uri = new Uri("file://DEMOXP/Documents/lorum.txt ");
if (uri.Scheme == Uri.UriSchemeFile)
{
    System.Net.FileWebRequest request =
        (System.Net.FileWebRequest)System.Net.FileWebRequest.Create(uri);
    System.Net.FileWebResponse response =
        (System.Net.FileWebResponse)request.GetResponse();
    System.IO.StreamReader reader =
        new System.IO.StreamReader(response.GetResponseStream());
    string tmp = reader.ReadToEnd();
    response.Close();
}
```

In this listing, we are requesting the `lorum.txt` file that exists in the `Documents` folder on the DEMOXP machine on our local network.

Sending Mail

Finally, consider a feature common to many Web applications — the capability to send e-mail from a Web page. The capability to send mail was part of the 1.0 Framework and located in the `System.Web.Mail` namespace. In the 2.0 Framework, this functionality was enhanced and moved to the `System.Net.Mail` namespace. Listing 25-29 shows an example of sending an e-mail.

Listing 25-29: Sending mail from a Web page

VB

```
Dim message As _
    New System.Net.Mail.MailMessage("webmaster@ineta.org", "webmaster@ineta.org")
message.Subject = "Sending Mail with ASP.NET 3.5"
message.Body = _
    "This is a sample email which demonstrates sending email using ASP.NET 3.5"

Dim smtp As New System.Net.Mail.SmtpClient("localhost")
smtp.Send(message)
```

C#

```
System.Net.Mail.MailMessage message =
    new System.Net.Mail.MailMessage("webmaster@ineta.org","webmaster@ineta.org");
message.Subject = "Sending Mail with ASP.NET 3.5";
message.Body =
    "This is a sample email which demonstrates sending email using ASP.NET 3.5";
System.Net.Mail.SmtpClient smtp = new System.Net.Mail.SmtpClient("localhost");
smtp.Send(message);
```

In this sample, you first create a `MailMessage` object, which is the class that contains the actual message you want to send. The `MailMessage` class requires the To and From address be provided to its constructor, and you can either provide the parameters as strings, or you can use the `MailAddressCollection` class to provide multiple recipients' e-mail addresses.

After you create the `Message`, you use the `SmtpClient` class to actually send the message to your local SMTP server. The `SmtpClient` class allows you to specify the SMTP Server from which you want to relay your e-mail.

Summary

In this chapter, you looked at some of the other classes in the .NET Framework. You looked at managing the local file system by using classes in the `System.IO` namespace such as `DirectoryInfo` and the `FileInfo`, and you learned how to enumerate the local file system and manipulate both directory and file properties and directory and file Access Control Lists. Additionally, the chapter discussed the rich functionality .NET provides for working with paths.

The chapter also covered how the .NET Framework enables you to read and write data to a multitude of data locations, including the local file system, network file system, and even system memory through a common `Stream` architecture. The Framework provides you with specialized classes to deal with each kind of data location. Additionally, the Framework makes working with streams even easier by providing `Reader` and `Writer` classes. These classes hide much of the complexity of reading from and writing to underlying streams. Here, too, the Framework provides you with a number of different `Reader` and `Writer` classes that give you the power to control exactly how your data is read or written, be it character, binary, string, or XML.

You were also introduced to a new feature of the .NET 2.0 Framework that allows you to communicate with serial ports.

Finally, you learned about the variety of network communication options the .NET Framework provides. From making and sending Web requests over HTTP, FTP, and File, to sending mail, the .NET Framework offers you a full plate of network communication services.

26

User and Server Controls

In an object-oriented environment like .NET, the encapsulation of code into small, single-purpose, reusable objects is one of the keys to developing a robust system. For instance, if your application deals with customers, you might want to consider creating a customer's object that encapsulates all the functionality a customer might need. The advantage is that you create a single point with which other objects can interact, and you have only a single point of code to create, debug, deploy, and maintain. In this scenario, the customer object is typically known as a business object because it encapsulates all the business logic needed for a customer.

Several other types of reusable objects are available in .NET. In this chapter, we concentrate on discussing and demonstrating how you can create reusable visual components for an ASP.NET application. The two types of reusable components in ASP.NET are user controls and server controls.

A *user control* encapsulates existing ASP.NET controls into a single container control, which you can easily reuse throughout your Web project.

A *server control* encapsulates the visual design, behavior, and logic for an element that the user interacts with on the Web page.

Visual Studio ships with a large number of server controls that you are probably already familiar with, such as the Label, Button, and TextBox controls. This chapter talks about how you can create custom server controls and extend existing server controls.

Because the topics of user controls and server controls are so large, and because discussing the intricacies of each could easily fill an entire book by itself, this chapter can't possibly investigate every option available to you. Instead, it attempts to give you a brief overview of building and using user controls and server controls and demonstrates some common scenarios for each type of control. By the end of this chapter, you should have learned enough that you can get started building basic controls of each type and be able to continue to learn on your own.

User Controls

User controls represent the simplest form of ASP.NET control encapsulation. Because they are the simplest, they are also the easiest to create and use. Essentially a *user control* is the grouping of existing server controls into a single-container control. This enables you to create powerful objects that you can easily use throughout an entire Web project.

Creating User Controls

Creating user controls is very simple in Visual Studio 2008. To create a new user control, you first add a new User Control file to your Web site. From the Website menu, select the Add New Item option. After the Add New File dialog appears, select the Web User Control File template from the list and click OK. Notice that after the file is added to the project, the file has an `.ascx` extension. This extension signals to ASP.NET that this file is a user control. If you attempt to load the user control directly into your browser, ASP.NET returns an error telling you that this type of file cannot be served to the client.

If you look at the HTML source (shown in Listing 26-1) for the user control, you see several interesting differences from a standard ASP.NET Web page.

Listing 26-1: A Web user control file template

```
<%@ Control Language="VB" ClassName="WebUserControl1" %>

<script runat="server">

</script>
```

First, notice that the source uses the `@Control` directive rather than the `@Page` directive, which a standard Web page would use. Second, notice that unlike a standard ASP.NET Web page, no other HTML tags besides the `<script>` tags exist in the control. The Web page containing the user control provides the basic HTML, such as the `<body>` and `<form>` tags. In fact, if you try to add a server-side form tag to the user control, ASP.NET returns an error when the page is served to the client. The error message tells you that only one server-side form tag is allowed in your Web page.

To add controls to the form, simply drag them from the Toolbox onto your user control. Listing 26-2 shows the user control after a Label and a Button have been added.

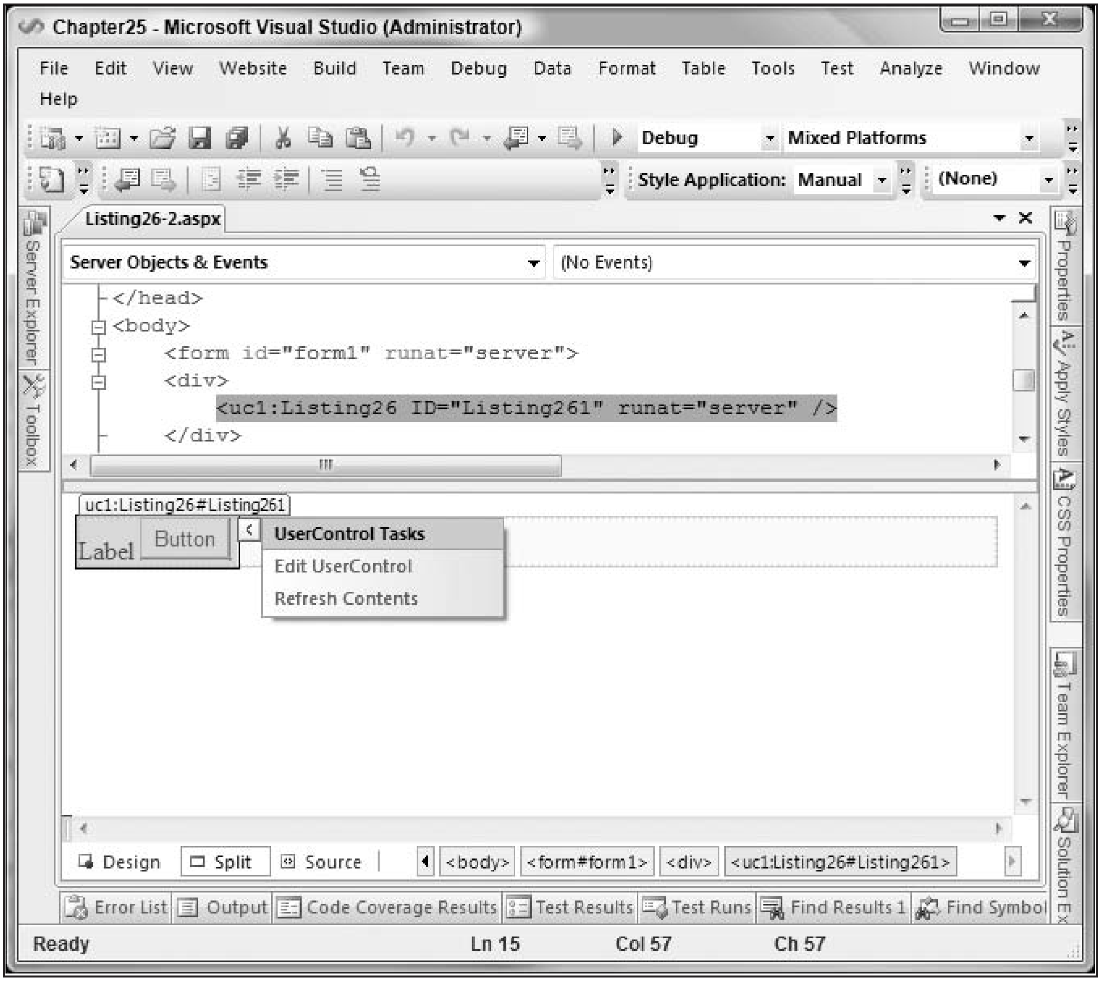
Listing 26-2: Adding controls to the Web user control

```
<%@ Control Language="VB" ClassName="WebUserControl2" %>

<script runat="server">

</script>

<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
<asp:Button ID="Button1" runat="server" Text="Button" />
```

After you add the controls to the user control, you put the user control onto a standard ASP.NET Web page. To do this, drag the file from the Solution Explorer onto your Web page.

> *If you are familiar with using user controls in prior versions of Visual Studio, you probably remember the gray control representation that appeared in the page designer when you dropped a user control onto a Web page. In Visual Studio 2005, this experience was improved and user controls are now fully rendered on the host Web page during design time. This allows you to see an accurate representation of what the entire page will look like after it is rendered to the client.*

Figure 26-1 shows the user control after it has been dropped onto a host Web page.


Chapter25 - Microsoft Visual Studio (Administrator)
File Edit View Website Build Team Debug Data Format Table Tools Test Analyze Window Help
Debug
Mixed Platforms
Style Application: Manual
(None)
Listing26-2.aspx
Server Objects & Events
(No Events)
uc1:Listing26#Listing261
Label
Button
UserControl Tasks
Edit UserControl
Refresh Contents
Design
Split
Source
<body>
<form#form1>
<div>
<uc1:Listing26#Listing261>
Server Explorer
Toolbox
Properties
Apply Styles
CSS Properties
Team Explorer
Solution Ex
Error List
Output
Code Coverage Results
Test Results
Test Runs
Find Results 1
Find Symbol
Ready
Ln 15
Col 57
Ch 57

```
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <uc1:Listing26 ID="Listing261" runat="server" />
    </div>
```

Figure 26-1

After you have placed the user control onto a Web page, open the page in a browser to see the fully rendered Web page.

User controls fully participate in the page-rendering lifecycle, and controls contained within a user control behave identically to controls placed onto a standard ASP.NET Web page. This means that the user control has its own page execute events (such as Init, Load, and Prerender) that execute as the page is processed. It also means that child control events, such as a button-click event, will behave identically. Listing 26-3 shows how to use the User Controls `Page_Load` event to populate the label and to handle the button-click event.

Listing 26-3: Creating control events in a user control

VB

```
<%@ Control Language="VB" ClassName="WebUserControl1" %>

<script runat="server">

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Me.Label1.Text = "The quick brown fox jumped over the lazy dog"
    End Sub

    Protected Sub Button1_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)
        Me.Label1.Text = "The quick brown fox clicked the button on the page"
    End Sub
</script>

<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click" />
```

C#

```
<%@ Control Language="C#" ClassName="WebUserControl1" %>

<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        this.Label1.Text = "The quick brown fox jumped over the lazy dog";
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        this.Label1.Text = "The quick brown fox clicked the button on the page";
    }
</script>
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click" />
```

Now when you render the Web page, you see that the text of the label changes as the user control loads, and again when you click the bottom of the page. In fact, if you put a breakpoint on either of these two events, you can see that ASP.NET does indeed break, even inside the user control code when the page is executed.

Interacting with User Controls

So far, you have learned how you can create user controls and add them to a Web page. You have also learned how user controls can execute their own code. Most user controls, however, are not islands on their parent page. Many scenarios require that the host Web page be able to interact with user controls that have been placed on it. For instance, you may decide that the text you want to load in the label must be given to the user control by the host page. To do this, you simply add a public property to the user control, and then assign text using the property. Listing 26-4 shows the modified user control.

Listing 26-4: Exposing user control properties

VB

```
<%@ Control Language="VB" ClassName="WebUserControl" %>

<script runat="server">

    Private _text As String

    Public Property Text() As String
        Get
            Return _text
        End Get
        Set(ByVal value As String)
            _text = value
        End Set
    End Property

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Me.Label1.Text = Me.Text
    End Sub

    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        Me.Label1.Text = "The quick brown fox clicked the button on the page"
    End Sub

</script>

<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click" />
```

C#

```
<%@ Control Language="C#" ClassName="WebUserControl" %>

<script runat="server">
    private string _text;

    public string Text
    {
        get {
            return _text;
        }
        set {
            _text = value;
        }
    }

    protected void Page_Load(object sender, EventArgs e)
    {
        this.Label1.Text = this.Text;
    }
```

Continued

```
    protected void Button1_Click(object sender, EventArgs e)
    {
        this.Label1.Text = "The quick brown fox clicked the button on the page";
    }

</script>

<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click" />
```

After you modify the user control, you simply populate the property from the host Web page. Listing 26-5 shows how to set the `Text` property in code, but public properties exposed by user controls will also be exposed by the Property Browser.

Listing 26-5: Populating user control properties from the host Web page

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Me.WebUserControl1.Text = "The quick brown fox jumped over the lazy dog"
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    this.WebUserControl1.Text = "The quick brown fox jumped over the lazy dog";
}
```

User controls are simple ways of creating powerful, reusable components in ASP.NET. They are easy to create using the built-in templates. Because they participate fully in the page lifecycle, you can create controls that can interact with their host page and even other controls on the host page.

Loading User Controls Dynamically

User controls can also be created and added to the Web form dynamically at runtime. The ASP.NET `Page` object includes the `LoadControl` method, which allows you to load user controls at runtime by providing the method with a virtual path to the user control you want to load. The method returns a generic `Control` object that you can then add to the page's `Controls` collection. Listing 26-6 demonstrates how you can use the `LoadControl` method to dynamically add a user control to a Web page.

Listing 26-6: Dynamically adding a user control

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim myForm As Control = Page.FindControl("form1")
        Dim c1 As Control = LoadControl("~/WebUserControl.ascx")
        myForm.Controls.Add(c1)
    End Sub
</script>
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    </div>
    </form>
</body>
</html>
```

C#

```
<script runat="server">
    void Page_Load(object sender, EventArgs e)
    {
        Control myForm = Page.FindControl("Form1");
        Control c1 = LoadControl("~/WebUserControl.ascx");
        myForm.Controls.Add(c1);
    }
</script>
```

The first step in adding a user control to the page is to locate the page's Form control using the `FindControl` method. Should the user control contain ASP.NET controls that render form elements such as a button or text box, this user control must be added to the Form element's `Controls` collection.

> *It is possible to add user controls containing certain ASP.NET elements such as a Label, HyperLink, or Image directly to the* `Page` *object's* `Controls` *collection; however, it is generally safer to be consistent and add them to the Form. Adding a control that must be contained within the Form, such as a Button control, to the* `Pages Controls` *collection results in a runtime parser error.*

After the form has been found, the sample uses the Page's `LoadControl()` method to load an instance of the user control. The method accepts a virtual path to the user control you want to load and returns the loaded user control as a generic `Control` object.

Finally, the control is added to the `Form` object's `Controls` collection. You can also add the user control to other container controls that may be present on the Web page, such as a Panel or Placeholder control.

> *Remember that you need to re-add your control to the ASP.NET page each time the page performs a postback.*

After you have the user control loaded, you can also work with its object model, just as you can with any other control. To access properties and methods that the user control exposes, you cast the control from the generic Control type to its actual type. To do that, you also need to add the `@Reference` directive to the page. This tells ASP.NET to compile the user control and link it to the ASP.NET page so that the page knows where to find the user control type. Listing 26-7 demonstrates how you can access a custom property of your user control by casting the control after loading it. The sample loads a modified user control that hosts an ASP.NET TextBox control and exposes a public property that allows you to access the TextBox control's `Text` property.

Listing 26-7: Casting a user control to its native type

VB

```
<%@ Page Language="VB" %>
<%@ Reference Control="~/WebUserControl.ascx" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim myForm As Control = Page.FindControl("form1")
        Dim c1 As WebUserControl = _
            CType(LoadControl("~/WebUserControl.ascx"), WebUserControl)
        myForm.Controls.Add(c1)

        c1.ID = "myWebUserControl1"
        c1.Text = "My users controls text"
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" %>
<%@ Reference Control="~/WebUserControl.ascx" %>
<script runat="server">
    void Page_Load(object sender, EventArgs e)
    {
        Control myForm = Page.FindControl("Form1");
        WebUserControl c1 =
            (WebUserControl)LoadControl("WebUserControl.ascx");
        myForm.Controls.Add(c1);

        c1.ID = "myWebUserControl1";
        c1.Text = "My users controls text";
    }
</script>
```

Notice that the sample adds the control to the Form's `Controls` collection and then sets the `Text` property. This ordering is actually quite important. After a page postback occurs the control's ViewState is not calculated until the control is added to the `Controls` collection. If you set the `Text` value (or any other property of the user control) before the control's ViewState, the value is not persisted in the ViewState. One twist to dynamically adding user controls occurs when you are also using Output Caching to cache the user controls. In this case, after the control has been cached, the `LoadControl` method does not return a new instance of the control. Instead, it returns the cached copy of the control. This presents problems when you try to cast the control to its native type because, after the control is cached, the `LoadControl` method returns it as a `PartialCachingControl` object rather than as its native type. Therefore, the cast in the previous sample results in an exception being thrown.

To solve this problem, you simply test the object type before attempting the cast. This is shown in Listing 26-8.

Listing 26-8: Detecting cached user controls

VB

```
<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim myForm As Control = Page.FindControl("form1")
        Dim c1 As Control = LoadControl("~/WebUserControl.ascx")
        myForm.Controls.Add(c1)
        If (c1.GetType() Is GetType(WebUserControl)) Then
            'This control is not participating in OutputCache
            CType(c1, WebUserControl).ID = "myWebUserControl1"
            CType(c1, WebUserControl).Text = "My users controls text"

        ElseIf (c1.GetType() Is GetType(PartialCachingControl) And _
            ((CType(c1, PartialCachingControl)).CachedControl IsNot Nothing)) Then

            'The control is participating in output cache, but has expired
            Dim myWebUserControl as WebUserControl = _
                CType((CType(c1, PartialCachingControl).CachedControl), _
                WebUserControl)

            myWebUserControl.ID = "myWebUserControl1"
            myWebUserControl.Text = "My users controls text"
        End If

    End Sub
</script>
```

C#

```
<script runat="server">
    void Page_Load(object sender, EventArgs e)
    {
        Control myForm = Page.FindControl("Form1");
        Control c1 = LoadControl("WebUserControl.ascx");
        myForm.Controls.Add(c1);

        if (c1 is WebUserControl)
        {
            //This control is not participating in OutputCache
            ((WebUserControl)c1).ID = "myWebUserControl1";
            ((WebUserControl)c1).Text = "My users controls text";
        }
        else if ((c1 is PartialCachingControl) &&
            ((PartialCachingControl)c1).CachedControl != null)
        {
            //The control is participating in output cache, but has expired
             WebUserControl myWebUserControl =
                ((WebUserControl)((PartialCachingControl)c1).CachedControl);
            myWebUserControl.ID = "myWebUserControl1";
            myWebUserControl.Text = "My users controls text";
        }
    }
</script>
```

The sample demonstrates how you can test to see what type the `LoadControl` returns and set properties based on the type. For more information on caching, check out Chapter 23.

Finally, in all the previous samples that demonstrate dynamically adding user controls, the user controls have been added during the `Page_Load` event. But there may be times when you want to add the control based on other events, such as a button's `Click` event or the `SelectedIndexChanged` event of a DropDownList control. Using these events to add user controls dynamically presents challenges. Specifically, because the events may not be raised each time a page postback occurs, you need to create a way to track when a user control has been added so that it can be re-added to the Web page as additional postbacks occur.

A simple way to do this is to use the ASP.NET session to track when the user control is added to the Web page. Listing 26-9 demonstrates this.

Listing 26-9: Tracking added user controls across postbacks

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
         'Make sure the control has not already been added to the page
         If ((Session("WebUserControlAdded") Is Nothing) Or _
             (Not CBool(Session("WebUserControlAdded")))) Then

             Dim myForm As Control = Page.FindControl("Form1")
             Dim c1 As Control = LoadControl("WebUserControl.ascx")
             myForm.Controls.Add(c1)

             Session("WebUserControlAdded") = True
         End If
     End Sub

     Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
         'Check to see if the control should be added to the page
         If ((Session("WebUserControlAdded") IsNot Nothing) And _
             (CBool(Session("WebUserControlAdded")))) Then

             Dim myForm As Control = Page.FindControl("Form1")
             Dim c1 As Control = LoadControl("WebUserControl.ascx")
             myForm.Controls.Add(c1)
         End If
     End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Button ID="Button1" runat="server" Text="Button"
            OnClick="Button1_Click" />
        <asp:Button ID="Button2" runat="server" Text="Button" />
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
```

```
        {
            //Make sure the control has not already been added to the page
            if ((Session["WebUserControlAdded"] == null) ||
                (!(bool)Session["WebUserControlAdded"]))
            {
                Control myForm = Page.FindControl("Form1");
                Control c1 = LoadControl("WebUserControl.ascx");
                myForm.Controls.Add(c1);

                Session["WebUserControlAdded"] = true;
            }
        }

        protected void Page_Load(object sender, EventArgs e)
        {
            //Check to see if the control should be added to the page
            if ((Session["WebUserControlAdded"] != null) &&
                ((bool)Session["WebUserControlAdded"]))
            {
                Control myForm = Page.FindControl("Form1");
                Control c1 = LoadControl("WebUserControl.ascx");
                myForm.Controls.Add(c1);
            }
        }
    </script>
```

This sample used a simple `Session` variable to track whether the user control has been added to the page. When the Button1 `Click` event fires, the session variable is set to `True`, indicating that the user control has been added. Then, each time the page performs a postback, the `Page_Load` event checks to see if the session variable is set to `True`, and if so, it re-adds the control to the page.

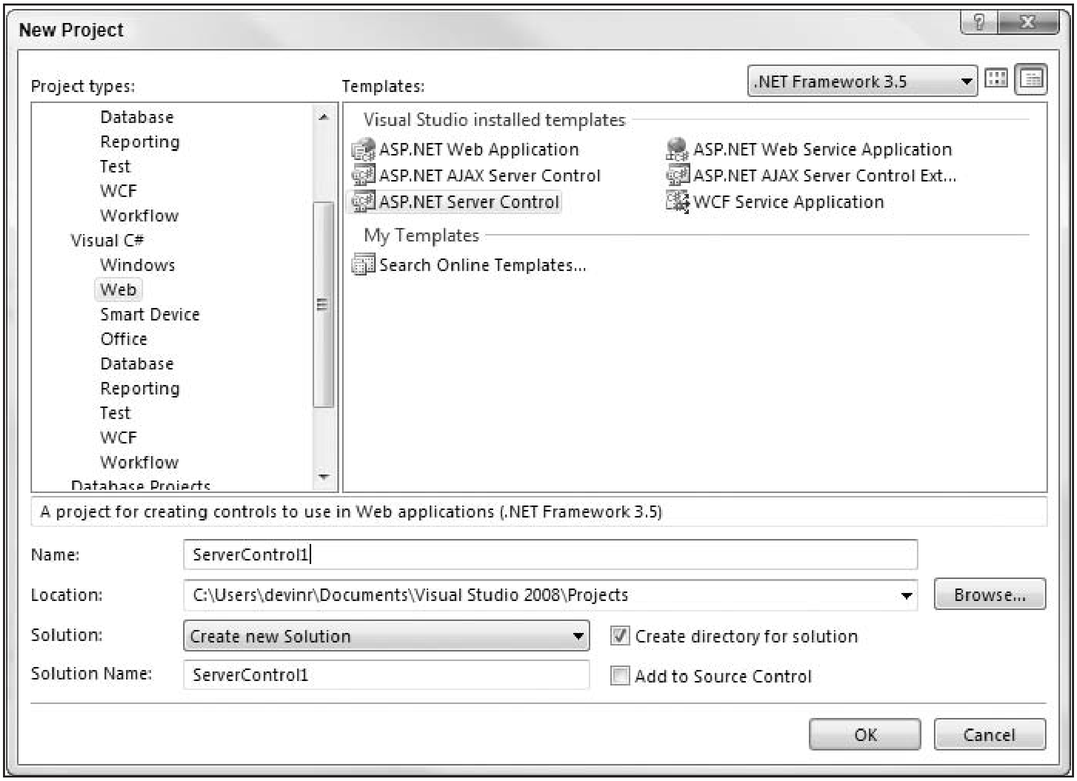
Server Controls

The power to create server controls in ASP.NET is one of the greatest tools you can have as an ASP.NET developer. Creating your own custom server controls and extending existing controls are actually both quite easy. In ASP.NET 3.5, all controls are derived from two basic classes: `System.Web.UI.WebControls.WebControl` or `System.Web.UI.ScriptControl`. Classes derived from the `WebControl` class have the basic functionality required to participate in the Page framework. These classes include most of the common functionality needed to create controls that render a visual HTML representation and provide support for many of the basic styling elements such as Font, Height, and Width. Because the `WebControl` class derives from the `Control` class, the controls derived from it have the basic functionality to be a designable control, meaning they can be added to the Visual Studio Toolbox, dragged onto the page designer, and have their properties and events displayed in the Property Browser.

Controls derived from the `ScriptControl` class build on the functionality that the `WebControl` class provides by including additional features designed to make working with client-side script libraries easier. The class tests to ensure that a ScriptManager control is present in the hosting page during the controls’ PreRender stage, and also ensures that derived controls call the the proper ScriptManager methods during the Render event.

WebControl Project Setup

This section demonstrates just how easy it is to create custom server controls by creating a very simple server control that derives from the `WebControl` class. In order to create a new server control, you create a new ASP.NET Server Controlproject. You can use this project to demonstrate concepts throughout the rest of this chapter. In Visual Studio, choose File ➪ New Project to open the New Project dialog box. From the Project Types tree, open either the Visual Basic or Visual C# nodes and select the Web node. Figure 26-2 shows the New Project dialog with a Visual C# ASP.NET Server Control project template selected.


New Project
Project types:
Database
Reporting
Test
WCF
Workflow
Visual C#
Windows
Web
Smart Device
Office
Database
Reporting
Test
WCF
Workflow
Database Projects
Templates:
.NET Framework 3.5
Visual Studio installed templates
ASP.NET Web Application
ASP.NET Web Service Application
ASP.NET AJAX Server Control
ASP.NET AJAX Server Control Ext...
ASP.NET Server Control
WCF Service Application
My Templates
Search Online Templates...
Browse...
OK
Cancel

A project for creating controls to use in Web applications (.NET Framework 3.5)

Name: ServerControl1

Location: C:\Users\devinr\Documents\Visual Studio 2008\Projects

Solution: Create new Solution

Solution Name: ServerControl1

☑ Create directory for solution

☐ Add to Source Control

Figure 26-2

When you click OK in the New Project dialog box, Visual Studio creates a new ASP.NET Server Control project for you. Notice that the project includes a template class that contains a very simple server control. Listing 26-10 shows the code for this template class.

Listing 26-10: The Visual Studio ASP.NET Server Control class template

VB

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModelImports System.Text
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls
```

```
<DefaultProperty("Text"), _
 ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")> _
Public Class ServerControl1
    Inherits WebControl

   <Bindable(True), Category("Appearance"), DefaultValue(""), Localizable(True)> _
    Property Text() As String
        Get
            Dim s As String = CStr(ViewState("Text"))
            If s Is Nothing Then
                Return "[" + Me.ID + "]"
            Else
                Return s
            End If
        End Get

        Set(ByVal Value As String)
            ViewState("Text") = Value
        End Set
    End Property

    Protected Overrides Sub RenderContents(ByVal output As HtmlTextWriter)
        output.Write(Text)
    End Sub

End Class
```

C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ServerControl1
{
    [DefaultProperty("Text")]
    [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")]
    public class ServerControl1 : WebControl
    {
        [Bindable(true)]
        [Category("Appearance")]
        [DefaultValue("")]
        [Localizable(true)]
        public string Text
        {
            get
            {
                String s = (String)ViewState["Text"];
                return ((s == null) ? "[" + this.ID + "]" : s);
            }
            set
            {
                ViewState["Text"] = value;
            }
        }

        protected override void RenderContents(HtmlTextWriter output)
        {
            output.Write(Text);
        }
    }
}
```

This template class creates a basic server control that exposes one property called `Text` and renders the value of that property to the screen. Notice that you override the `RenderContents` method of the control and write the value of the `Text` property to the pages output stream. We talk more about rendering output later in the chapter.

> *Note that creating a server control project is not the only way to create an ASP.NET server control. Visual Studio 2008 also provides a basic ASP.NET Server Control file template that you can add to either an existing ASP.NET Server Control project, or to any other standard class library project. This template however differs slightly from the template used to create the default server control included in the ASP.NET Server control project. It uses a different filename scheme, and includes slightly different code in the default Text propertys getter.*

Now, take this class and use it in a sample Web application by adding a new Web Project to the existing solution. The default Web page, created by Visual Studio, serves as a test page for the server control samples in this chapter.

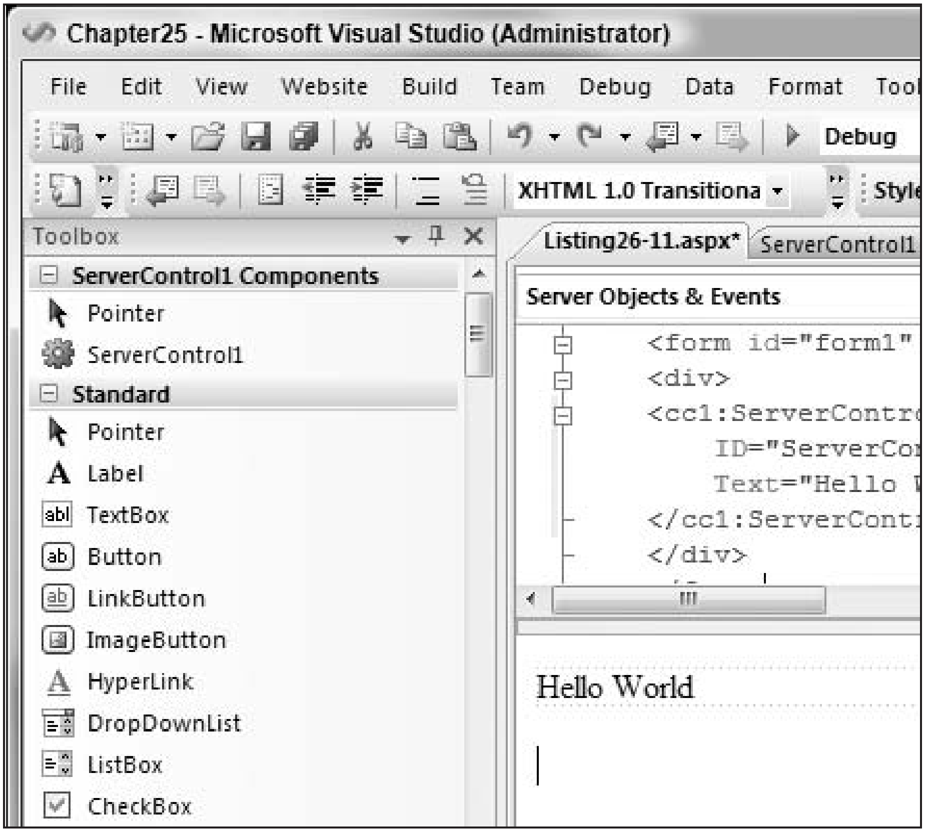
Visual Studio 2008 will automatically add any controls contained in projects in the open Solution to the Toolbox for you. To see this, simply build the Solution and then open the default Web page of the Web Project you just added. The Toolbox should contain a new section called WebControlLibrary1 .Components, and the new server control should be listed in this section (see Figure 26-3).

Now, all you have to do is drag the control onto the Web Form, and the control's assembly is automatically added to the Web project for you. When you drag the control from the Toolbox onto the designer surface, the control adds itself to the Web page. Listing 26-11 shows you what the Web page source code looks like after you have added the control.

Listing 26-11: Adding a Web Control Library to a Web page

```
<%@ Register Assembly="MyServerControl"
    Namespace="ServerControl1" TagPrefix="cc1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Adding a Custom Web Control</title>
</head>
```

```
<body>
    <form id="form1" runat="server">
    <div>
        <cc1:ServerControl1 ID="ServerControl1" runat="server" />
    </div>
    </form>
</body>
</html>
```


Chapter25 - Microsoft Visual Studio (Administrator)
File
Edit
View
Website
Build
Team
Debug
Data
Format
Tool
Debug
XHTML 1.0 Transitiona
Style
Toolbox
ServerControl1 Components
Pointer
ServerControl1
Standard
Pointer
Label
TextBox
Button
LinkButton
ImageButton
HyperLink
DropDownList
ListBox
CheckBox
Listing26-11.aspx*
ServerControl1
Server Objects & Events

```
    <form id="form1"
    <div>
    <cc1:ServerContr
        ID="ServerCo
        Text="Hello
    </cc1:ServerCont
    </div>
```

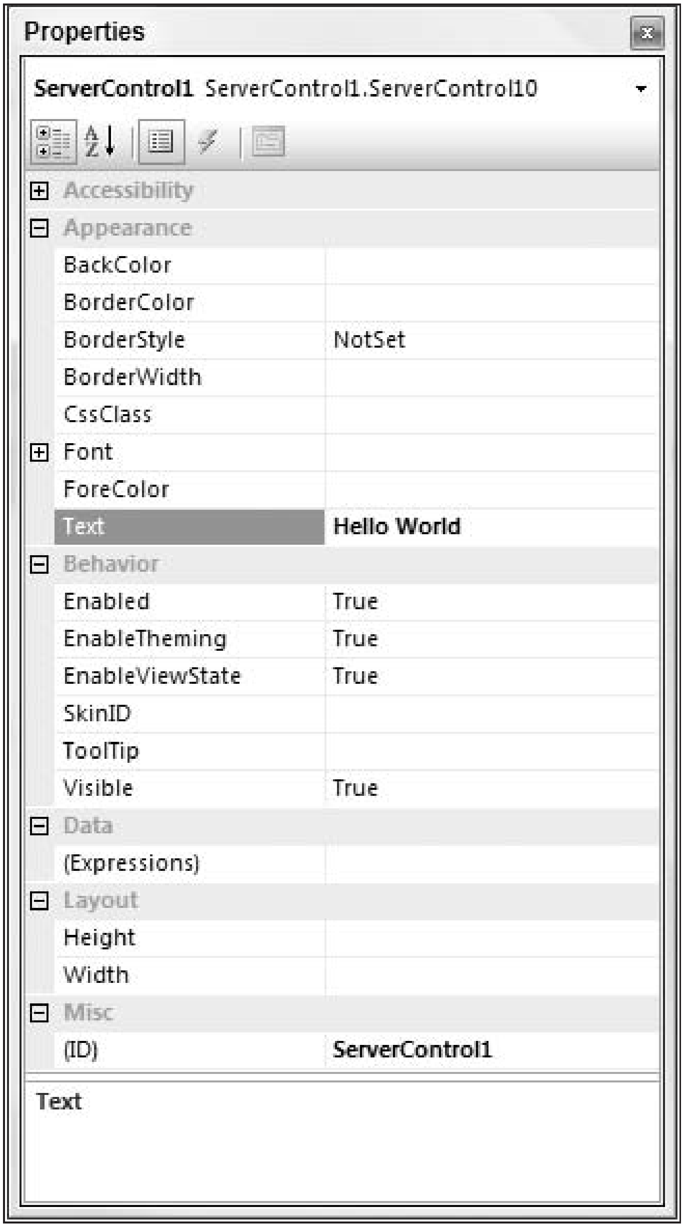
Hello World

Figure 26-3

After you drag the control onto the Web form, take a look at its properties in the Properties Window. Figure 26-4 shows the properties of your custom control.

Notice that in addition to the `Text` property you defined in the control, the control has all the basic properties of a visual control, including various styling and behavior properties. The properties are exposed because the control was derived from the `WebControl` class. The control also inherits the base events exposed by `WebControl`.

Make sure the control is working by entering a value for the `Text` property and viewing the page in a browser. Figure 26-5 shows what the page looks like if you set the `Text` property to `"Hello World!"`.



Properties

ServerControl1 ServerControl1.ServerControl10

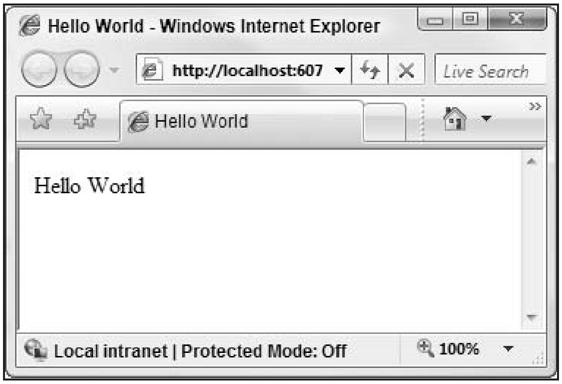
⊞ Accessibility	
⊟ Appearance	
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	
CssClass	
⊞ Font	
ForeColor	
Text	**Hello World**
⊟ Behavior	
Enabled	True
EnableTheming	True
EnableViewState	True
SkinID	
ToolTip	
Visible	True
⊟ Data	
(Expressions)	
⊟ Layout	
Height	
Width	
⊟ Misc	
(ID)	**ServerControl1**

Text

Figure 26-4

As expected, the control has rendered the value of the `Text` property to the Web page. If you view the HTML source of this sample, you will see that not only has ASP.NET added the value of the Text property to the HTML markup, but it has surrounded the text with a `<SPAN>` block. If you look at the code for the `WebControl` class's render method, you can see why.

```
protected internal override void Render(HtmlTextWriter writer)
{
    this.RenderBeginTag(writer);
    this.RenderContents(writer);
    this.RenderEndTag(writer);
}
```


Hello World - Windows Internet Explorer
http://localhost:607
Live Search
Hello World
Local intranet | Protected Mode: Off
100%

Hello World

Figure 26-5

You can see that, by default, the `Render` method, which calls the `RenderContents` method, also includes the `RenderBeginTag` and `RenderEndTag` methods, causing the `Span` tags to be added. If you have provided an `ID` value for your control, then the `Span` tag will also, by default, render an `ID` attribute. Having the `Span` tags can sometimes be problematic, so if you want to prevent this in ASP.NET, you simply override the `RenderMethod` and call the `RenderContents` method directly.

VB

```
    Protected Overrides Sub Render(ByVal writer As System.Web.UI.HtmlTextWriter)
        Me.RenderContents(writer)
    End Sub
```

C#

```
protected override void Render(HtmlTextWriter writer)
{
    this.RenderContents(writer);
}
```

This prevents ASP.NET from automatically adding the `Span` tags.

The samples in this section demonstrate just how easy it is to create a simple server control. Of course, this control does not have much functionality and lacks many of the features of a server control. The following section shows how you can use attributes to enhance this server control to make it more useful and user-friendly.

Control Attributes

A key enhancement to the design-time experience for users utilizing server controls is achieved by adding attributes to the class level and to the control's classes and properties. Attributes define much of how the control behaves at design time in Visual Studio. For instance, when you look at the default control template from the previous section (Listing 26-6), notice that attributes have been applied to both the `Class` and to the `Text` property. In this section, you study these attributes and how they affect the behavior of the control.

Class Attributes

Class attributes generally control how the server control behaves in the Visual Studio Toolbox and when placed on the design surface. The class attributes can be divided into three basic categories: attributes that help the Visual Studio designer know how to render the control at design time, attributes that help you tell ASP.NET how to render nested controls, and attributes that tell Visual Studio how to display the control in the Toolbox. The following table describes some of these attributes.

Attribute	Description
`Designer`	Indicates the designer class this control should use to render a design-time view of the control on the Visual Studio design surface
`TypeConverter`	Specifies what type to use as a converter for the object
`DefaultEvent`	Indicates the default event created when the user double-clicks the control on the Visual Studio design surface
`DefaultProperty`	Indicates the default property for the control
`ControlBuilder`	Specifies a `ControlBuilder` class for building a custom control in the ASP.NET control parser
`ParseChildren`	Indicates whether XML elements nested within the server controls tags will be treated as properties or as child controls
`TagPrefix`	Indicates the text the control is prefixed with in the Web page HTML

Property/Event Attributes

Property attributes are used to control a number of different aspects of server controls. You can use attributes to control how your properties and events behave in the Visual Studio Property Browser. You can also use attributes to control how properties and events are serialized at design time. The following table describes some of the property and event attributes you can use.

Obviously, the class and property/event attribute tables present a lot of information upfront. You already saw a demonstration of some of these attributes in Listing 26-1; now, as you go through the rest of the chapter, you will spend time working with most of the attributes listed in the tables.

Control Rendering

Now that that you have seen the large number of options you have for working with a server control at design-time, look at what you need to know to manage how your server control renders its HTML at runtime.

Attribute	Description
`Bindable`	Indicates that the property can be bound to a data source
`Browsable`	Indicates whether the property should be displayed at design time in the Property Browser
`Category`	Indicates the category this property should be displayed under in the Property Browser
`Description`	Displays a text string at the bottom of the Property Browser that describes the purpose of the property
`EditorBrowsable`	Indicates whether the property should be editable when shown in the Property Browser
`DefaultValue`	Indicates the default value of the property shown in the Property Browser
`DesignerSerializationVisibility`	Specifies the visibility a property has to the design-time serializer
`NotifyParentProperty`	Indicates that the parent property is notified when the value of the property is modified
`PersistChildren`	Indicates whether, at design-time, the child controls of a server control should be persisted as nested inner controls
`PersistanceMode`	Specifies how a property or an event is persisted to the ASP.NET page
`TemplateContainer`	Specifies the type of INamingContainer that will contain the template once it is created
`Editor`	Indicates the UI Type Editor class this control should use to edit its value
`Localizable`	Indicates that the property contains text that can be localized
`Themable`	Indicates whether this property can have a theme applied to it

The Page Event Lifecycle

Before we talk about rendering HTML, you must understand the lifecycle of a Web page. As the control developer, you are responsible for overriding methods that execute during the lifecycle and implementing your own custom rendering logic.

Remember that when a Web browser makes a request to the server, it is using HTTP, a stateless protocol. ASP.NET provides a page-execution framework that helps create the illusion of state in a Web application. This framework is basically a series of methods and events that execute every time an

ASP.NET page is processed. You may have seen diagrams showing this lifecycle for ASP.NET 1.0. Since ASP.NET 2.0, a variety of additional events have been available to give you more power over the behavior of the control. Figure 26-6 shows the events and methods called during the control's lifecycle.

Many events and members are executed during the control's lifecycle, but you should concentrate on the more important among them.

Rendering Services

The main job of a server control is to render some type of markup language to the HTTP output stream, which is returned to and displayed by the client. If your client is a standard browser, the control should emit HTML; if the client is something like a mobile device, the control may need to emit a different type of markup, such as WAP, or WML. As I stated earlier, it is your responsibility as the control developer to tell the server control what markup to render. The overridden `RenderContents` method, called during the control's lifecycle, is the primary location where you tell the control what you want to emit to the client. In Listing 26-12, notice that the `RenderContents` method is used to tell the control to print the value of the `Text`property.

Listing 26-12: Overriding the Render method

VB

```
Protected Overrides Sub RenderContents(ByVal output As HtmlTextWriter)
    output.Write(Text)
End Sub
```

C#

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.Write(Text);
}
```

Also notice that the `RenderContents` method has one method parameter called `output`. This parameter is an `HtmlTextWriter` class, which is what the control uses to render HTML to the client. This special writer class is specifically designed to emit HTML 4.0–compliant HTML to the browser. The `HtmlTextwriter` class has a number of methods you can use to emit your HTML, including `RenderBeginTag` and `WriteBeginTag`. Listing 26-13 shows how you can modify the control's `Render` method to emit an HTML `<input>` tag.

Listing 26-13: Using the HtmlTextWriter to render an HTML tag

VB

```
Protected Overrides Sub RenderContents(ByVal output As HtmlTextWriter)
    output.RenderBeginTag(HtmlTextWriterTag.Input)
    output.RenderEndTag()
End Sub
```

C#

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.RenderBeginTag(HtmlTextWriterTag.Input);
    output.RenderEndTag();
}
```

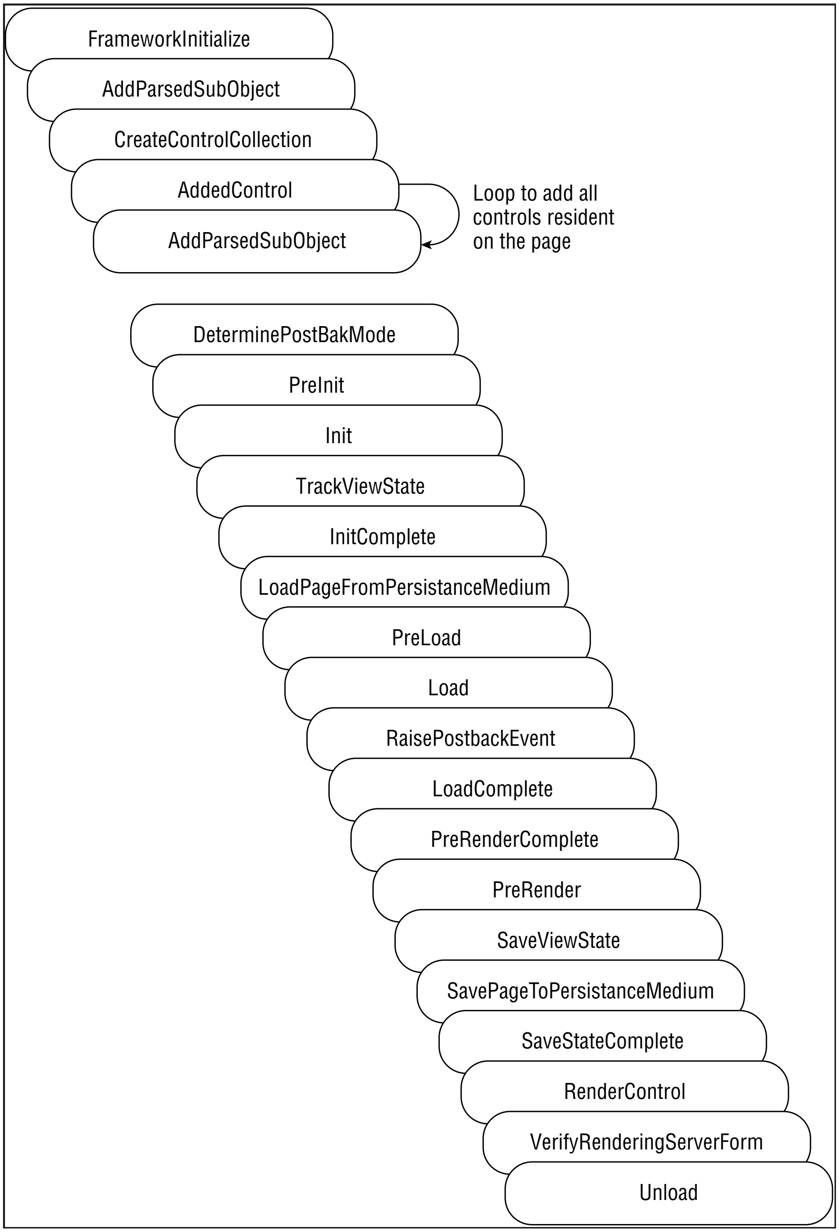

FrameworkInitialize
AddParsedSubObject
CreateControlCollection
AddedControl
AddParsedSubObject
Loop to add all controls resident on the page
DeterminePostBakMode
PreInit
Init
TrackViewState
InitComplete
LoadPageFromPersistanceMedium
PreLoad
Load
RaisePostbackEvent
LoadComplete
PreRenderComplete
PreRender
SaveViewState
SavePageToPersistanceMedium
SaveStateComplete
RenderControl
VerifyRenderingServerForm
Unload

Figure 26-6

First, notice that the `RenderBeginTag` method is used to emit the HTML. The advantage of using this method to emit HTML is that it requires you to select a tag from the `HtmlTextWriterTag` enumeration. Using the `RenderBeginTag` method and the `HtmlTextWriterTag` enumeration enables you to have your control automatically support downlevel browsers that cannot understand HTML 4.0 syntax. If a downlevel browser is detected by ASP.NET, the control automatically emits HTML 3.2 syntax instead of HTML 4.0.

Second, notice that the `RenderEndTag` method is also used. As the name suggests, this method renders the closing tag. Notice, however, that you do not have to specify in this method which tag you want to close. The `RenderEndTag` automatically closes the last begin tag rendered by the `RenderBeginTag` method, which in this case is the `<input>` tag. If you want to emit multiple HTML tags, make sure you order your `Begin` and `End` render methods properly. In Listing 26-14, for example, you add a `<div>` tag to the control. The `<div>` tag surrounds the `<input>` tag when rendered to the page.

Listing 26-14: Using the HtmlTextWriter to render multiple HTML tags

VB

```
Protected Overrides Sub RenderContents(ByVal output As HtmlTextWriter)
    output.RenderBeginTag(HtmlTextWriterTag.Div)
    output.RenderBeginTag(HtmlTextWriterTag.Input)
    output.RenderEndTag()
    output.RenderEndTag()
End Sub
```

C#

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.RenderBeginTag(HtmlTextWriterTag.Div);
    output.RenderBeginTag(HtmlTextWriterTag.Input);
    output.RenderEndTag();
    output.RenderEndTag();
}
```

Now that you have a basic understanding of how to emit simple HTML, look at the output of your control. You can do this by viewing the test HTML page containing the control in a browser and choosing View ➪ Source. Figure 26-7 shows the source for the page.

You can see that the control emitted some pretty simple HTML markup. Also notice (in the highlighted area) that the control was smart enough to realize that the input control did not contain any child controls and, therefore, the control did not need to render a full closing tag. Instead, it automatically rendered the shorthand `/ >`, rather than `</input>`.

Adding Tag Attributes

Emitting HTML tags is a good start to building the control, but perhaps this is a bit simplistic. Normally, when rendering HTML you would emit some tag attributes (such as `ID` or `Name`) to the client in addition to the tag. Listing 26-15 shows how you can easily add tag attributes.

Listing 26-15: Rendering HTML tag attributes

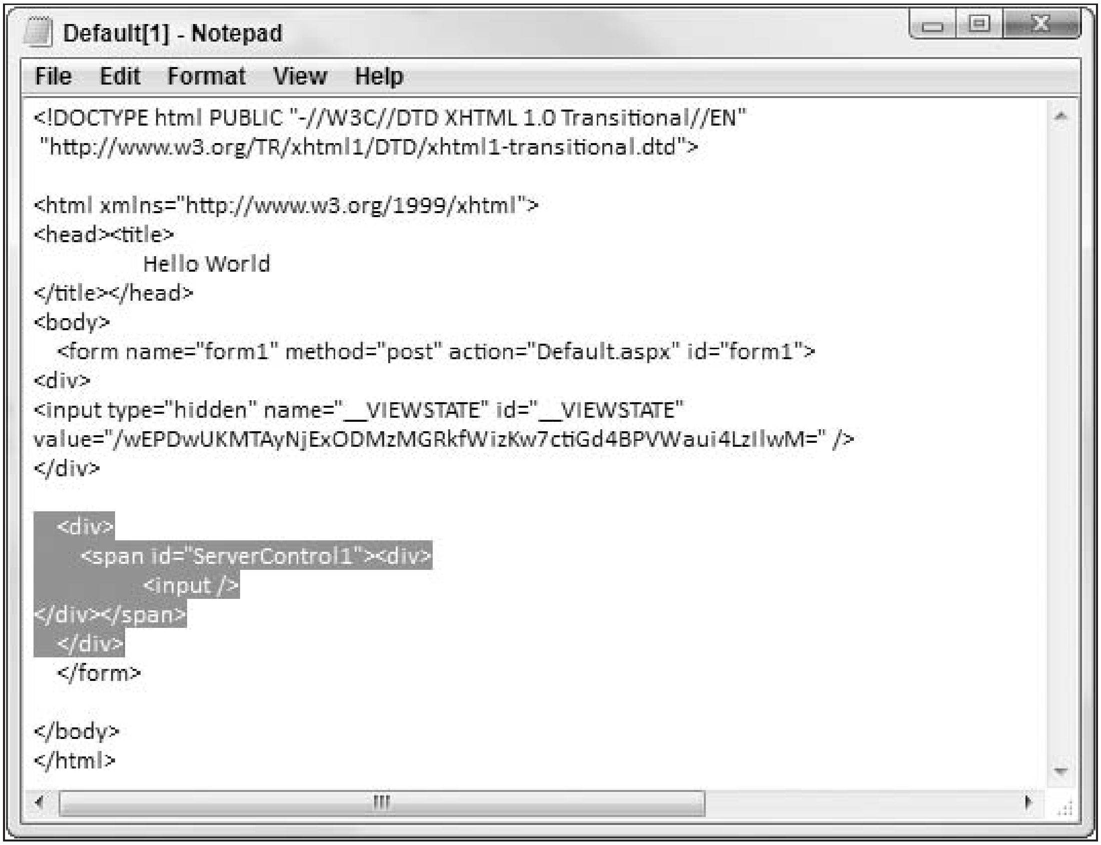
VB

```
Protected Overrides Sub RenderContents(ByVal output As HtmlTextWriter)
    output.RenderBeginTag(HtmlTextWriterTag.Div)

    output.AddAttribute(HtmlTextWriterAttribute.Type, "text")
    output.AddAttribute(HtmlTextWriterAttribute.Id, Me.ClientID & "_i")
    output.AddAttribute(HtmlTextWriterAttribute.Name, Me.ClientID & "_i")
    output.AddAttribute(HtmlTextWriterAttribute.Value, Me.Text)
```

```
        output.RenderBeginTag(HtmlTextWriterTag.Input)
        output.RenderEndTag()

        output.RenderEndTag()
    End Sub
```

C#

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.RenderBeginTag(HtmlTextWriterTag.Div);

    output.AddAttribute(HtmlTextWriterAttribute.Type, "text");
    output.AddAttribute(HtmlTextWriterAttribute.Id, this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Name, this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Value, this.Text);
    output.RenderBeginTag(HtmlTextWriterTag.Input);
    output.RenderEndTag();

    output.RenderEndTag();
}
```



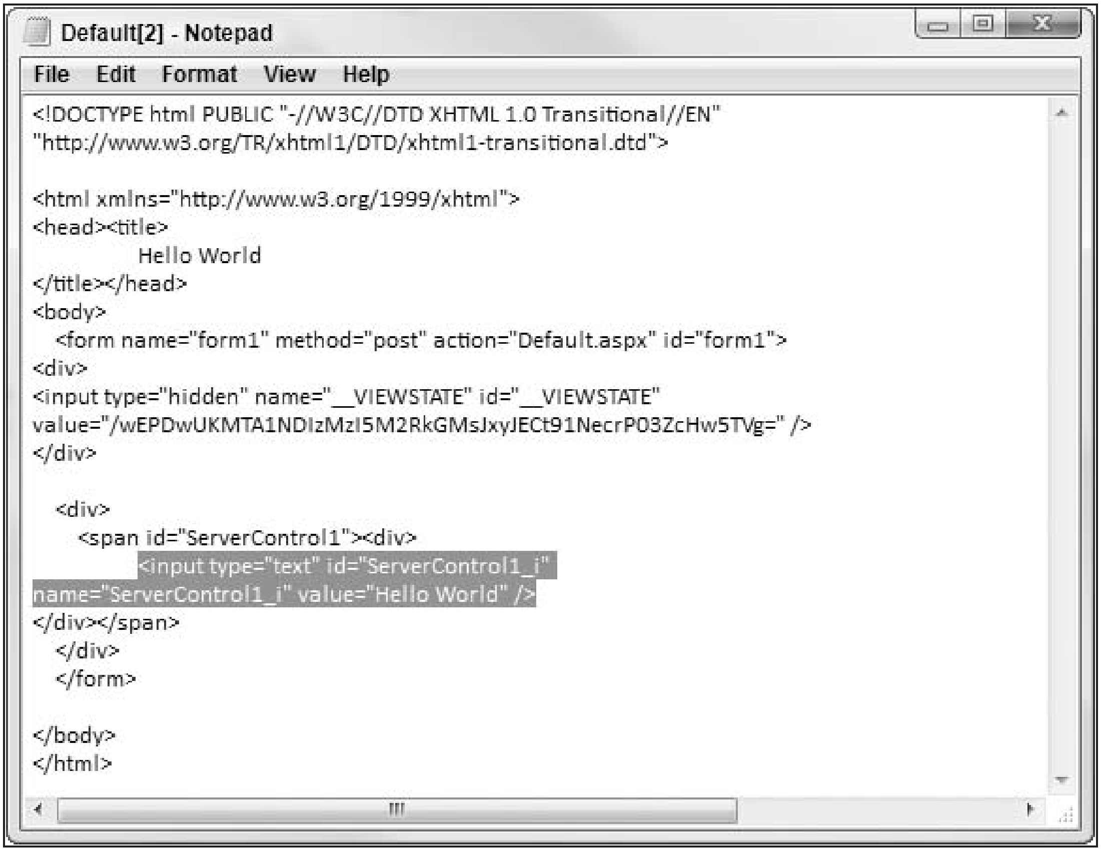
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
            Hello World
</title></head>
<body>
  <form name="form1" method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUKMTAyNjExODMzMGRkfWizKw7ctiGd4BPVWaui4LzIlwM=" />
</div>

  <div>
    <span id="ServerControl1"><div>
          <input />
</div></span>
  </div>
  </form>

</body>
</html>
```

Figure 26-7

You can see that by using the `AddAttribute` method, you have added three attributes to the `<input>` tag. Also notice that, once again, you are using an enumeration, `HtmlTextWriterAttribute`, to select the attribute you want to add to the tag. This serves the same purpose as using the `HtmlTextWriterTag` enumeration, allowing the control to degrade its output to downlevel browsers.

As with the `Render` methods, the order in which you place the `AddAttributes` methods is important. You place the `AddAttributes` methods directly before the `RenderBeginTag` method in the code. The `AddAttributes` method associates the attributes with the next HTML tag that is rendered by the `RenderBeginTag` method — in this case the `<input>` tag.

Now browse to the test page and check out the HTML source with the added tag attributes. Figure 26-8 shows the HTML source rendered by the control.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
            Hello World
</title></head>
<body>
   <form name="form1" method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUKMTA1NDIzMzI5M2RkGMsJxyJECt91NecrP03ZcHw5TVg=" />
</div>

   <div>
      <span id="ServerControl1"><div>
            <input type="text" id="ServerControl1_i"
name="ServerControl1_i" value="Hello World" />
</div></span>
   </div>
   </form>

</body>
</html>
```

Figure 26-8

You can see that the tag attributes you added in the server control are now included as part of the HTML tag rendered by the control.

A Word about Control IDs

Notice that in Listing 26-11 it's important to use the control's `ClientID` property as the value of both the `Id`and `Name` attributes. Controls that derive from the `WebControl` class automatically expose three different types of ID properties: `ID`, `UniqueID`, and `ClientID`. Each of these properties exposes a slightly altered version of the control's ID for use in a specific scenario.

The `ID` property is the most obvious. Developers use it to get and set the control's ID. It must be unique to the page at design time.

The `UniqueID` property is a read-only property generated at runtime that returns an ID that has been prepended with the containing control's ID. This is essential so that ASP.NET can uniquely identify each control in the page's control tree, even if the control is used multiple times by a container control such as a Repeater or GridView. For example, if you add this custom control to a repeater, the `UniqueID` for each custom control rendered by the Repeater is modified to include the Repeater's ID when the page executed:

```
MyRepeater:Ctrl0:MyCustomControl
```

The `ClientID` property is essentially identical to the `UniqueID` property with one important exception. The `ClientID` property always uses an underscore (_) to separate the ID values, rather than using the value of the `IdSeparator` property. This is because the ECMAScript standard disallows the use of colons in ID attribute values, which is the default value of the `IdSeparator` property. Using the underscore ensures that a control can be used by client-side JavaScript.

Additionally, in order to ensure that controls can generate a unique ID, they should implement the INamingContainer interface. This is a marker interface only, meaning that it does not require any additional methods to be implemented; it does, however, ensure that the ASP.NET runtime guarantees the control always has a unique name within the page's tree hierarchy, regardless of its container.

Styling HTML

So far, you have seen how easy it is to build a simple HTML control and emit the proper HTML, including attributes. In this section, we discuss how you can have your control render style information. As mentioned at the very beginning of this section, you are creating controls that inherit from the `WebControl` class. Because of this, these controls already have the basic infrastructure for emitting most of the standard CSS-style attributes. In the Property Browser for this control, you should see a number of style properties already listed, such as background color, border width, and font. You can also launch the style builder to create complex CSS styles. These basic properties are provided by the `WebControl` class, but it is up to you to tell your control to render the values set at design time. To do this, you simply execute the `AddAttributesToRender` method. Listing 26-16 shows you how to do this.

Listing 26-16: Rendering style properties

VB

```
Protected Overrides Sub RenderContents(ByVal output As HtmlTextWriter)
   output.RenderBeginTag(HtmlTextWriterTag.Div)

    output.AddAttribute(HtmlTextWriterAttribute.Type, "text")
    output.AddAttribute(HtmlTextWriterAttribute.Id, Me.ClientID & "_i")
    output.AddAttribute(HtmlTextWriterAttribute.Name, Me.ClientID & "_i")
    output.AddAttribute(HtmlTextWriterAttribute.Value, Me.Text)
    Me.AddAttributesToRender(output)

    output.RenderBeginTag(HtmlTextWriterTag.Input)
    output.RenderEndTag()

    output.RenderEndTag()
End Sub
```

C#

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.RenderBeginTag(HtmlTextWriterTag.Div);

    output.AddAttribute(HtmlTextWriterAttribute.Type, "text");
    output.AddAttribute(HtmlTextWriterAttribute.Id, this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Name, this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Value, this.Text);
    this.AddAttributesToRender(output);

    output.RenderBeginTag(HtmlTextWriterTag.Input);
    output.RenderEndTag();

    output.RenderEndTag();
}
```

Executing this method tells the control to render any style information that has been set.

Note that executing this method not only causes the style-related properties to be rendered, but also several other attributes, including `ID`, `tabindex`, and `tooltip`. If you are manually rendering these attributes earlier in your control, then you may end up with duplicate attributes being rendered. Additionally, be careful about where you use the `AddAttributesToRender` method. In Listing 26-26, it is executed immediately before the `Input` tag is rendered, which means that the attributes will be rendered both on the `Input` element and on the `Span` element surrounding the Input element.

Try placing the method call before the beginning DIV tag is rendered, or after the DIV’s end tag is rendered. You will see that in the case of the former, the attributes are now applied to the DIV and its surrounding span and in the case of the latter, only the SPAN has the attribute applied.

Using the Property Browser, you can set the background color of the control to `Red` and the font to `Bold`. When you set these properties, they are automatically added to the control tag in the ASP.NET page. After you have added the styles, the control tag looks like this:

```
<cc1:ServerControl1 BackColor="Red" Font-Bold=true
 ID="ServerControl11" runat="server" />
```

The style changes have been persisted to the control as attributes. When you execute this page in the browser, the style information should be rendered to the HTML, making the background of the text box red and its font bold. Figure 26-9 shows the page in the browser.

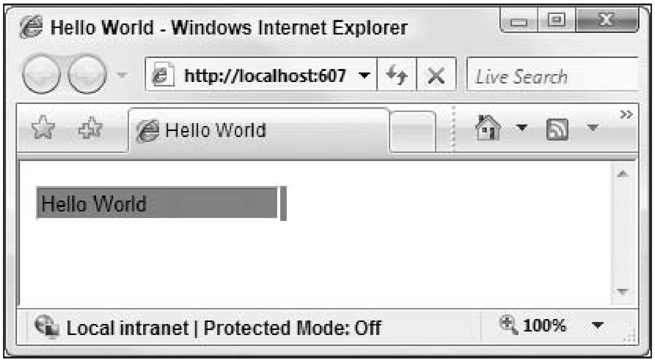
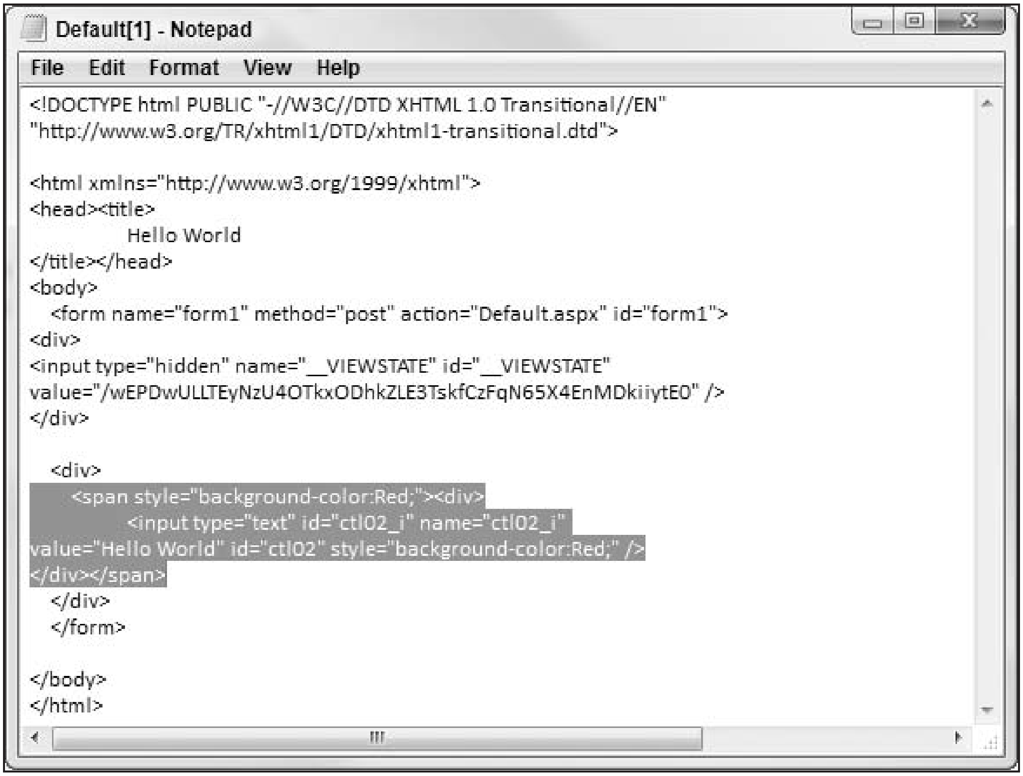

Hello World - Windows Internet Explorer
http://localhost:607
Live Search
Hello World
Hello World
Local intranet | Protected Mode: Off
100%

Figure 26-9

Once again, look at the source for this page. The style information has been rendered to the HTML as a style tag. Figure 26-10 shows the HTML emitted by the control.


Default[1] - Notepad
File Edit Format View Help

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
        Hello World
</title></head>
<body>
  <form name="form1" method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwULLTEyNzU4OTkxODhkZLE3TskfCzFqN65X4EnMDkiiytE0" />
</div>

  <div>
    <span style="background-color:Red;"><div>
        <input type="text" id="ctl02_i" name="ctl02_i"
value="Hello World" id="ctl02" style="background-color:Red;" />
</div></span>
  </div>
  </form>

</body>
</html>
```

Figure 26-10

If you want more control over the rendering of styles in your control you can use the HtmlTextWriters `AddStyleAttribute` method. Similar to the `AddAttribute` method, the `AddStyleAttribute` method enables you to specify CSS attributes to add to a control using the `HtmlTextWriterStyle` enumeration. However, unlike the `AddAttribute` method, attributes added using `AddStyleAttribute` are defined inside of a style attribute on the control. Listing 26-17 demonstrates the use of the `AddStyleAttribute` method.

Listing 26-17: Adding control styes using AddStyleAttribute

VB

```
Protected Overrides Sub RenderContents(ByVal output As HtmlTextWriter)
    output.RenderBeginTag(HtmlTextWriterTag.Div)

    output.AddAttribute(HtmlTextWriterAttribute.Type, "text")
    output.AddAttribute(HtmlTextWriterAttribute.Id, Me.ClientID & "_i")
    output.AddAttribute(HtmlTextWriterAttribute.Name, Me.ClientID & "_i")
    output.AddAttribute(HtmlTextWriterAttribute.Value, Me.Text)

    output.AddStyleAttribute(HtmlTextWriterStyle.BackgroundColor, "Red")

    output.RenderBeginTag(HtmlTextWriterTag.Input)
    output.RenderEndTag()

    output.RenderEndTag()
End Sub
```

C#

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.RenderBeginTag(HtmlTextWriterTag.Div);

    output.AddAttribute(HtmlTextWriterAttribute.Type, "text");
    output.AddAttribute(HtmlTextWriterAttribute.Id, this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Name, this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Value, this.Text);

    output.AddStyleAttribute(HtmlTextWriterStyle.BackgroundColor, "Red");

    output.RenderBeginTag(HtmlTextWriterTag.Input);
    output.RenderEndTag();

    output.RenderEndTag();
}
```

Running this sample will result in the same red background color being applied to the control.

Themes and Skins

A great feature added in ASP.NET 2.0, and introduced to you in Chapter 6, is themes and skins. This feature allows you to create visual styles for your Web applications. In this section, you learn what you need to know about themes and skins when creating a server control.

As you saw in Chapter 6, skins are essentially a way to set default values for the UI elements of controls in your Web application. You simply define the control and its properties in a `.skin` file and the values are applied to the control at runtime. Listing 26-18 shows a sample skin.

Listing 26-18: Sample ASP.NET skin

```
<%@ Register Assembly="WebControlLibrary1" Namespace="WebControlLibrary1"
    TagPrefix="cc1" %>

<cc1:webcustomcontrol1 BackColor="Green" runat="server" />
```

By default, ASP.NET allows all control properties to be defined in the skin file, but obviously this is not always appropriate. Most exposed properties are non-UI related; therefore, you do not apply a theme to them. By setting the `Themeable` attribute to `False` on each of these properties, you prevent the application of a theme. Listing 26-19 shows how to do this in your control by disabling themes on the `Text` property.

Listing 26-19: Disabling theme support on a control property

VB

```
<Bindable(True), Category("Appearance"), DefaultValue(""), _
Localizable(True), Themeable(False)> _
Property Text() As String
    Get
        Dim s As String = CStr(ViewState("Text"))
        If s Is Nothing Then
            Return "[" + Me.ID + "]"
        Else
            Return s
        End If
    End Get

    Set(ByVal Value As String)
        ViewState("Text") = Value
    End Set
End Property
```

C#

```
[Bindable(true)]
[Category("Appearance")]
[DefaultValue("")]
[Localizable(true)]
[Themeable(false)]
public string Text
    {
        get
        {
            String s = (String)ViewState["Text"];
            return ((s == null) ? "["+ this.ID + "]": s);
        }
        set
        {
            ViewState["Text"] = value;
        }
    }
```

Now, if a developer attempts to define this property in his skin file, he receives a compiler error when the page is executed.

Adding Client-Side Features

Although the capability to render and style HTML is quite powerful by itself, other resources can be sent to the client, such as client-side scripts, images, and resource strings. ASP.NET 3.5 provides you with some powerful new tools for using client-side scripts in your server controls and retrieving other resources to the client along with the HTML your control emits. Additionally, ASP.NET now includes an entire model that allows you to make asynchronous callbacks from your Web page to the server.

Emitting Client-Side Script

Having your control emit client-side script such as VBScript or JavaScript enables you to add powerful client-side functionality to your control. Client-side scripting languages take advantage of the client's browser to create more flexible and easy-to-use controls. Although ASP.NET 1.0 provided some simple methods to emit client-side script to the browser, ASP.NET has since enhanced these capabilities and now provides a wide variety of methods for emitting client-side script that you can use to control where and how your script is rendered.

If you have already used ASP.NET 1.0 to render client-side script to the client, you are probably familiar with a few methods such as the `Page.RegisterClientScriptBlock` and the `Page.RegisterStartupScript` methods. Since ASP.NET 2.0, these classes have been deprecated. Instead, ASP.NET now uses the `ClientScriptManager` class, which you can access using `Page.ClientScript`. This class exposes various static client-script rendering methods that you can use to render client-side script.

Listing 26-20 demonstrates how you can use the `RegisterStartupScriptMethod` method to render JavaScript to the client. This listing adds the code into the `OnPreRender` method, rather than into the `Render` method used in previous samples. This method allows every control to inform the page about the client-side script it needs to render. After the `Render` method is called, the page is able to render all the client-side script it collected during the `OnPreRender` method. If you call the client-side script registration methods in the `Render` method, the page has already completed a portion of its rendering before your client-side script can render itself.

Listing 26-20: Rendering a client-side script to the browser

VB

```
Protected Overrides Sub OnPreRender(ByVal e As System.EventArgs)
    Page.ClientScript.RegisterStartupScript(GetType(Page), _
       "ControlFocus", "document.getElementById('" & Me.ClientID & _
           "_i" & "').focus();", _
       True)
End Sub
```

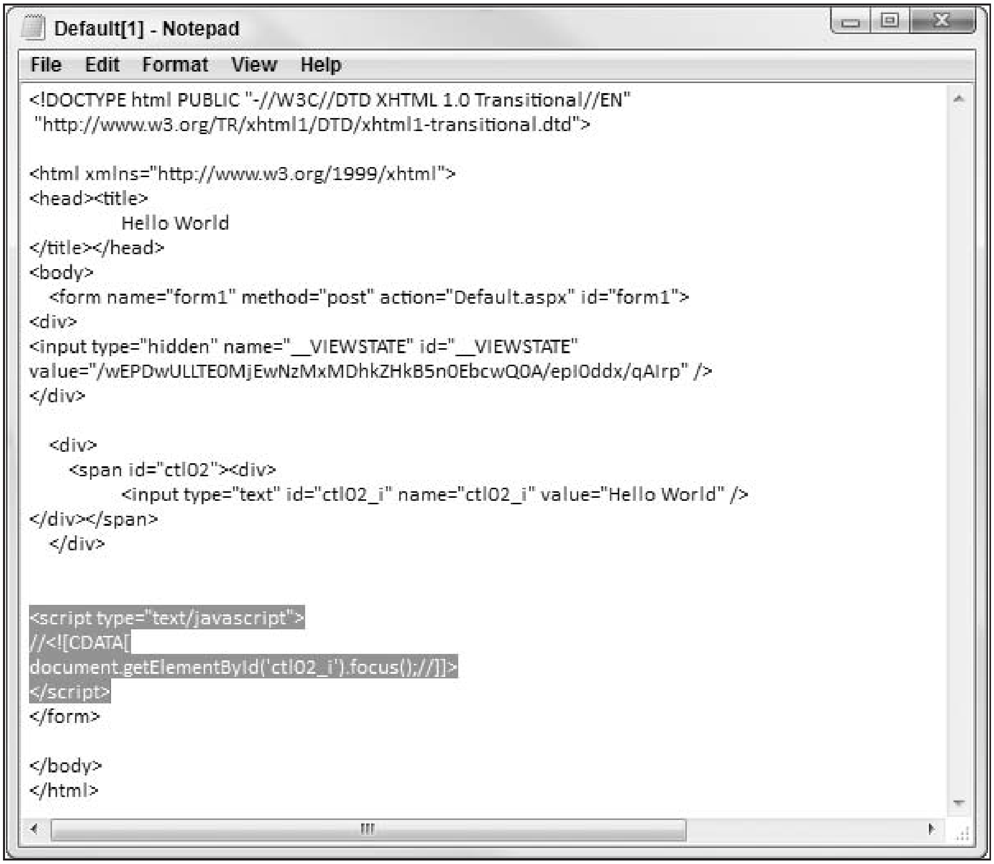
C#

```
protected override void OnPreRender(EventArgs e)
{
    Page.ClientScript.RegisterStartupScript( typeof(Page),
        "ControlFocus","document.getElementById('" + this.ClientID +
            "_i" + "').focus();",
        true);
}
```

In this listing, the code emits client-side script to automatically move the control focus to the TextBox control when the Web page loads. When you use the `RegisterStartupScript` method, notice that it now includes an overload that lets you specify if the method should render surrounding script tags. This can be handy if you are rendering more than one script to the page.

Also notice that the method requires a key parameter. This parameter is used to uniquely identify the script block; if you are registering more than one script block in the Web page, make sure that each block is supplied a unique key. You can use the `IsStartupScriptRegistered` method and the key to determine if a particular script block has been previously registered on the client using the `RegisterStatupScript` method.

When you execute the page in the browser, notice that the focus is automatically placed into a text box. If you look at the source code for the Web page, you should see that the JavaScript was written to the bottom of the page, as shown in Figure 26-11.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
        Hello World
</title></head>
<body>
   <form name="form1" method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwULLTE0MjEwNzMxMDhkZHkB5n0EbcwQ0A/epIOddx/qAIrp" />
</div>

   <div>
      <span id="ctl02"><div>
         <input type="text" id="ctl02_i" name="ctl02_i" value="Hello World" />
</div></span>
   </div>


<script type="text/javascript">
//<![CDATA[
document.getElementById('ctl02_i').focus();//]]>
</script>
</form>

</body>
</html>
```

Figure 26-11

If you want the script to be rendered to the top of the page, you use the `RegisterClientScriptBlock` method that emits the script block immediately after the opening `<form>` element.

Keep in mind that the browser parses the Web page from top to bottom, so if you emit client-side script at the top of the page that is not contained in a function, any references in that code to HTML elements further down the page will fail. The browser has not parsed that portion of the page yet.

Being able to render script that automatically executes when the page loads is nice, but it is more likely that you will want the code to execute based on an event fired from an HTML element on your page, such as the Click, Focus, or Blur events. In order to do this, you add an attribute to the HTML element you want the event to fire from. Listing 26-21 shows you how you can modify your control's `Render` and `PreRender` methods to add this attribute.

Listing 26-21: Using client-side script and event attributes to validate data

VB

```
Protected Overrides Sub RenderContents(ByVal output As HtmlTextWriter)
    output.RenderBeginTag(HtmlTextWriterTag.Div)

    output.AddAttribute(HtmlTextWriterAttribute.Type, "text")
    output.AddAttribute(HtmlTextWriterAttribute.Id, Me.ClientID & "_i")
    output.AddAttribute(HtmlTextWriterAttribute.Name, Me.ClientID & "_i")
    output.AddAttribute(HtmlTextWriterAttribute.Value, Me.Text)

    output.AddAttribute("OnBlur", "ValidateText(this)")

    output.RenderBeginTag(HtmlTextWriterTag.Input)
    output.RenderEndTag()

    output.RenderEndTag()

End Sub

Protected Overrides Sub OnPreRender(ByVal e As System.EventArgs)
    Page.ClientScript.RegisterStartupScript(GetType(Page), _
        "ControlFocus", "document.getElementById("' & Me.ClientID & _
            "_i" & "').focus();", _
       True)

    Page.ClientScript.RegisterClientScriptBlock( _
        GetType(Page), _
        "ValidateControl", _
        "function ValidateText() {" & _
            "if (ctl.value==") {" & _
                "alert('Please enter a value.');ctl.focus();} " & _
        "}", _
        True)
End Sub
```

C#

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.RenderBeginTag(HtmlTextWriterTag.Div);
    output.AddAttribute(HtmlTextWriterAttribute.Type, "text");
    output.AddAttribute(HtmlTextWriterAttribute.Id, this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Name, this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Value, this.Text);

    output.AddAttribute("OnBlur", "ValidateText(this)");

    output.RenderBeginTag(HtmlTextWriterTag.Input);
    output.RenderEndTag();
```

```
        output.RenderEndTag();
    }

    protected override void OnPreRender(EventArgs e)
    {
        Page.ClientScript.RegisterStartupScript(
            typeof(Page),
            "ControlFocus","document.getElementById('" + this.ClientID +
                "_i" + "').focus();",
            true);

        Page.ClientScript.RegisterClientScriptBlock(
            typeof(Page),
            "ValidateControl",
            "function ValidateText(ctl) {" +
                "if (ctl.value=='') {" +
                    "alert('Please enter a value.');ctl.focus();} " +
            "}",
            true);
    }
```

As you can see, the TextBox control is modified to check for an empty string. We have also included an attribute that adds the JavaScript `OnBlur` event to the text box. The `OnBlur` event fires when the control loses focus. When this happens, the client-side `ValidateText` method is executed, which we rendered to the client using `RegisterClientScriptBlock`.

The rendered HTML is shown in Figure 26-12.

Embedding JavaScript in the page is powerful, but if you are writing large amounts of client-side code, you might want to consider storing the JavaScript in an external file. You can include this file in your HTML by using the `RegisterClientScriptInclude` method. This method renders a script tag using the URL you provide to it as the value of its `src` element.

```
<script src="[url]" type="text/javascript"></script>
```

Listing 26-22 shows how you can modify the validation added to the TextBox control in Listing 26-20; but this time, the JavaScript validation function is stored in an external file.

Listing 26-22: Adding client-side script include files to a Web page

VB

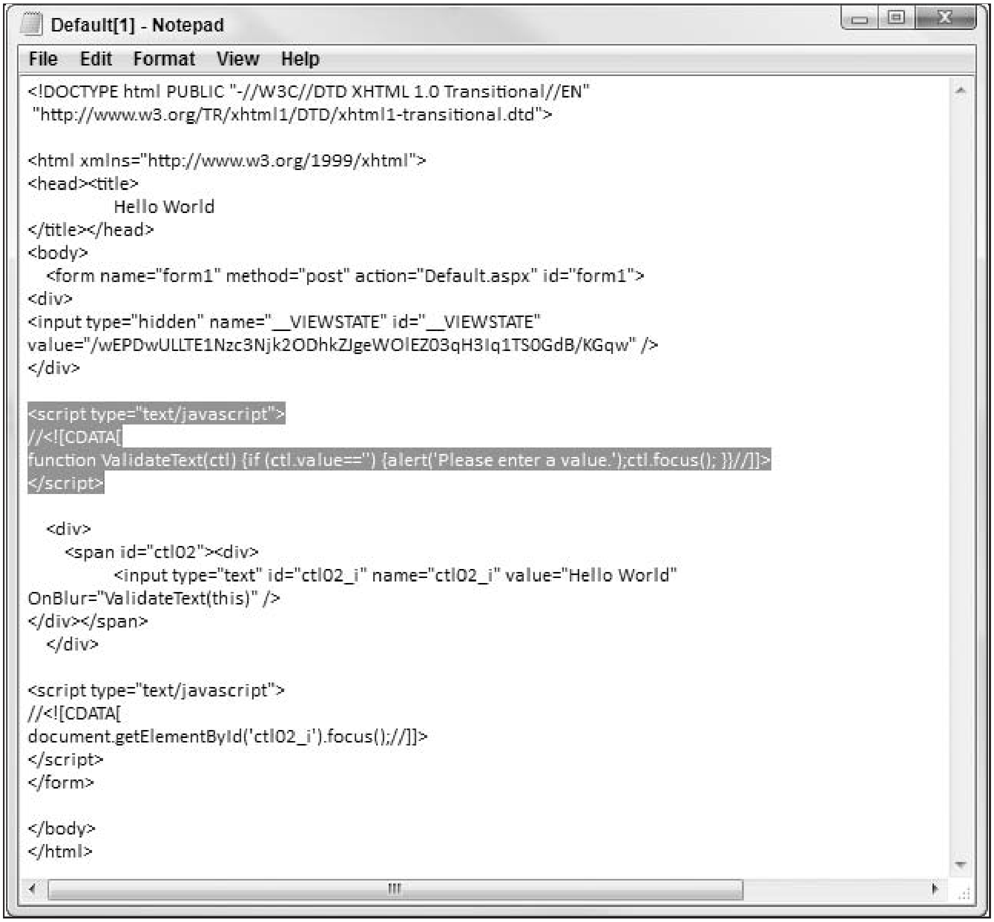
```
Protected Overrides Sub OnPreRender(ByVal e As System.EventArgs)
    Page.ClientScript.RegisterClientScriptInclude( _
       "UtilityFunctions", "JScript.js")

    Page.ClientScript.RegisterStartupScript(GetType(Page), _
        "ControlFocus", "document.getElementById('" & Me.ClientID & _
            "_i" & "').focus();", _
       True)
End Sub
```

C#

```
protected override void OnPreRender(EventArgs e)
{
```

Continued

```
        Page.ClientScript.RegisterClientScriptInclude(
            "UtilityFunctions", "JScript.js");

        Page.ClientScript.RegisterStartupScript(
            typeof(Page),
            "ControlFocus","document.getElementById('" + this.ClientID +
                "_i" + "').focus();",
            true);
    }
```



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
        Hello World
</title></head>
<body>
  <form name="form1" method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwULLTE1Nzc3Njk2ODhkZJgeWOlEZ03qH3Iq1TS0GdB/KGqw" />
</div>

<script type="text/javascript">
//<![CDATA[
function ValidateText(ctl) {if (ctl.value=='') {alert('Please enter a value.');ctl.focus(); }}//]]>
</script>

  <div>
    <span id="ctl02"><div>
        <input type="text" id="ctl02_i" name="ctl02_i" value="Hello World"
OnBlur="ValidateText(this)" />
</div></span>
  </div>

<script type="text/javascript">
//<![CDATA[
document.getElementById('ctl02_i').focus();//]]>
</script>
</form>

</body>
</html>
```

Figure 26-12

You have modified the `OnPreRender` event to register a client-side script file include, which contains the `ValidateText` function. You need to add a JScript file to the project and create the `ValidateText` function, as shown in Listing 26-23.

Listing 26-23: The validation JavaScript contained in the Jscript file

```
// JScript File

function ValidateText(ctl)
{
   if (ctl.value==") {
        alert('Please enter a value.');
        ctl.focus();
    }
}
```

The `ClientScriptManager` also provides methods for registering hidden HTML fields and adding script functions to the `OnSubmit` event.

Accessing Embedded Resources

A great way to distribute application resources like JavaScript files, images, or resource files is to embed them directly into the compiled assembly. While this was possible in ASP.NET 1.0, it was very difficult to access these resources as part of the page request process. ASP.NET 2.0 solved this problem by including the `RegisterClientScriptResource` method as part of the ClientScriptManager.

This method makes it possible for your Web pages to retrieve stored resources — like JavaScript files — from the compiled assembly at runtime. It works by using an HttpHandler to retrieve the requested resource from the assembly and return it to the client. The `RegisterClientScriptResource` method emits a `<script>` block whose `src` value points to this HttpHandler:

```
<script
     language="javascript"
     src="WebResource.axd?a=s&r=WebUIValidation.js&t=631944362841472848"
     type="text/javascript">
</script>
```

As you can see, the `WebResource.axd` handler is used to return the resource — in this case, the JavaScript file. You can use this method to retrieve any resource stored in the assembly, such as images or localized content strings from resource files.

Asynchronous Callbacks

Finally, ASP.NET also includes a convenient mechanism for enabling basic AJAX behavior, or client-side callbacks, in a server control. Client-side callbacks enable you to take advantage of the XmlHttp components found in most modern browsers to communicate with the server without actually performing a complete postback. Figure 26-13 shows how client-side callbacks work in the ASP.NET Framework.

In order to enable callbacks in your server control, you implement the `System.Web.UI.ICallBackEventHander` interface. This interface requires you to implement two methods, the `RaiseCallbackEvent` method and the `GetCallbackResult method`. These are the server-side events that fire when the client executes the callback. After you implement the interface, you want to tie your client-side events back to the server. You do this by using the `Page.ClientScript.GetCallbackEventReference` method.

This method allows you to specify the two client-side functions: one to serve as the callback handler and one to serve as an error handler. Listing 26-24 demonstrates how you can modify the TextBox control's `Render` methods and add the `RaiseCallBackEvent` method to use callbacks to perform validation.

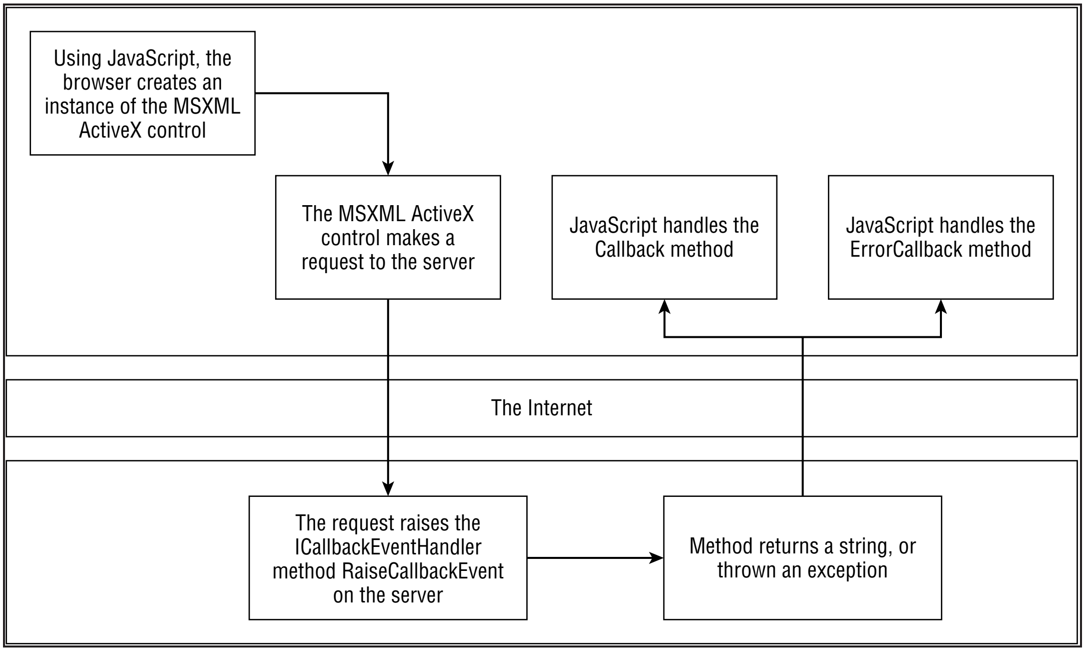

Using JavaScript, the browser creates an instance of the MSXML ActiveX control
The MSXML ActiveX control makes a request to the server
JavaScript handles the Callback method
JavaScript handles the ErrorCallback method
The Internet
The request raises the ICallbackEventHandler method RaiseCallbackEvent on the server
Method returns a string, or thrown an exception

Figure 26-13

Listing 26-24: Adding an asynchronous callback to validate data

VB

```
Protected Overrides Sub RenderContents(ByVal output As HtmlTextWriter)
    output.RenderBeginTag(HtmlTextWriterTag.Div)

    output.AddAttribute(HtmlTextWriterAttribute.Type, "text")
    output.AddAttribute(HtmlTextWriterAttribute.Id, Me.ClientID & "_i")
    output.AddAttribute(HtmlTextWriterAttribute.Name, Me.ClientID & "_i")
    output.AddAttribute(HtmlTextWriterAttribute.Value, Me.Text)

    output.AddAttribute("OnBlur", "ClientCallback();")
    Me.AddAttributesToRender(output)

    output.RenderBeginTag(HtmlTextWriterTag.Input)
    output.RenderEndTag()

    output.RenderEndTag()

End Sub

Protected Overrides Sub OnPreRender(ByVal e As System.EventArgs)
    Page.ClientScript.RegisterStartupScript(GetType(Page), _
        "ControlFocus", "document.getElementById('" & _
            Me.ClientID & "_i" & "').focus();", _
       True)

    Page.ClientScript.RegisterStartupScript( _
        GetType(Page), "ClientCallback", _
        "function ClientCallback() {" & _
            "args=document.getElementById('" & Me.ClientID & "_i" & "').value;" & _
            Page.ClientScript.GetCallbackEventReference(Me, "args", _
                "CallbackHandler", Nothing, "ErrorHandler", True) + "}", _
        True)
End Sub

Public Sub RaiseCallbackEvent(ByVal eventArgument As String) _
    Implements System.Web.UI.ICallbackEventHandler.RaiseCallbackEvent

    Dim result As Int32
    If (Not Int32.TryParse(eventArgument, result)) Then
        Throw New Exception("The method or operation is not implemented.")
    End If
End Sub

Public Function GetCallbackResult() As String _
    Implements System.Web.UI.ICallBackEventHandler.GetCallbackResult

    Return "Valid Data"
End Function
```

C#

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.RenderBeginTag(HtmlTextWriterTag.Div);

    output.AddAttribute(HtmlTextWriterAttribute.Type, "text");
    output.AddAttribute(HtmlTextWriterAttribute.Id, this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Name, this.ClientID + "_i");
    output.AddAttribute(HtmlTextWriterAttribute.Value, this.Text);

    output.AddAttribute("OnBlur", "ClientCallback();");
    this.AddAttributesToRender(output);

    output.RenderBeginTag(HtmlTextWriterTag.Input);
    output.RenderEndTag();

    output.RenderEndTag();
}
protected override void OnPreRender(EventArgs e)
{
    Page.ClientScript.RegisterStartupScript(
        typeof(Page),
        "ControlFocus","document.getElementById('" +
            this.ClientID + "_i" + "').focus();",
        true);
        Page.ClientScript.RegisterStartupScript(
            typeof(Page), "ClientCallback",
            "function ClientCallback() {" +
                "args=document.getElementById("' + this.ClientID + "_i" + "').value;" +
                Page.ClientScript.GetCallbackEventReference(this, "args",
                    "CallbackHandler", null,"ErrorHandler",true) + "{",
            true);
    }

    #region ICallbackEventHandler Members

    public void RaiseCallbackEvent(string eventArgument)
    {
        int result;
        if (!Int32.TryParse(eventArgument,out result) )
            throw new Exception("The method or operation is not implemented.");
    }

    public string GetCallbackResult()
    {
        return "Valid Data";
    }

    #endregion
```

As you can see, the `OnBlur` attribute has again been modified, this time by simply calling the `ClientCallback` method. This method is created and rendered during the `PreRender` event. The main purpose of this event is to populate the client-side `args` variable and call the client-side callback method.

You are using the `GetCallbackEventReference` method to generate the client-side script that actually initiates the callback. The parameters passed to the method indicate which control is initiating the callback, the names of the client-side callback method, and the name of the callback method parameters. The following table provides more details on the `GetCallbackEventReference` arguments.

Parameter	**Description**
`Control`	Server control that initiates the callback.
`Argument`	Client-side variable used to pass arguments to the server-side event handler.
`ClientCallback`	Client-side function serving as the `Callback`method. This method fires when the server-side processing has completed successfully.
`Context`	Client-side variable that gets passed directly to the receiving client-side function. The context does not get passed to the server.
`ClientErrorCallback`	Client-side function serving as the `Callback` error-handler method. This method fires when the server-side processing encounters an error.

In the code, two client-side methods are called: `CallbackHandler` and `ErrorHandler`, respectively. The two method parameters are `args` and `ctx`.

In addition to the server control code changes, the two client-side callback methods have been added to the JavaScript file. Listing 26-25 shows these new functions.

Listing 26-25: The client-side callback JavaScript functions

```
// JScript File
var args;
var ctx;

function ValidateText(ctl)
{
    if (ctl.value==") {
        alert('Please enter a value.');
        ctl.focus();
     }
}

function CallbackHandler(args,ctx)
{
    alert("The data is valid");
}

function ErrorHandler(args,ctx)
{
    alert("Please enter a number");
}
```

Now, when you view your Web page in the browser, as soon as the text box loses focus, you perform a client-side callback to validate the data. The callback raises the `RaiseCallbackEvent` method on the server, which validates the value of the text box that was passed to it in the `eventArguments`. If the value is valid, you return a string and the client-side `CallbackHandler` function fires. If the value is invalid, you throw an exception, which causes the client-side `ErrorHandler` function to execute.

Detecting and Reacting to Browser Capabilities

So far in the chapter we have described many powerful features, such as styling and emitting client-side scripts, that you can utilize when writing your own custom control. But if you are taking advantage of these features, you must also consider how you can handle certain browsers, often called downlevel browsers, that might not understand these advanced features or might not have them enabled. Being able to detect and react to downlevel browsers is an important consideration when creating your control. ASP.NET includes some powerful tools you can use to detect the type and version of the browser making the page request, as well as what capabilities the browser supports.

.browser files

ASP.NET 2.0 introduced a new and highly flexible method for configuring, storing, and discovering browser capabilities. All browser identification and capability information is now stored in `.browser` files. ASP.NET stores these files in the `C:\Windows\Microsoft.NET\Framework\v2.0.[xxxx]\CONFIG\Browsers` directory. If you open this folder, you see that ASP.NET provides you with a variety of `.browser` files that describe the capabilities of most of today's common desktop browsers, as well as information on browsers in devices such as PDAs and cellular phones. Open one of the browser files, and you see that the file contains all the identification and capability information for the browser. Listing 26-26 shows you the contents of the WebTV capabilities file.

Listing 26-26: A sample browser capabilities file

```
<browsers>
    <!-- sample UA "Mozilla/3.0 WebTV/1.2(Compatible;MSIE 2.0)" -->
    <browser id="WebTV" parentID="IE2">
        <identification>
            <userAgent
           match="WebTV/(?'version'(?'major'\d+)(?'minor'\.\d+)(?'letters'\w*))" />
        </identification>

        <capture>
        </capture>

        <capabilities>
            <capability name="backgroundsounds"    value="true" />
            <capability name="browser"             value="WebTV" />
            <capability name="cookies"             value="true" />
            <capability name="isMobileDevice"      value="true" />
            <capability name="letters"             value="\${letters}" />
            <capability name="majorversion"        value="\${major}" />
            <capability name="minorversion"        value="\${minor}" />
            <capability name="tables"              value="true" />
            <capability name="type"                value="WebTV\${major}" />
            <capability name="version"             value="\${version}" />
        </capabilities>

        <controlAdapters markupTextWriterType="System.Web.UI.Html32TextWriter">
    </controlAdapters>
    </browser>

    <browser id="WebTV2" parentID="WebTV">
        <identification>
            <capability name="minorversion" match="2" />
        </identification>

        <capture>
        </capture>

        <capabilities>
            <capability name="css1"                value="true" />
            <capability name="ecmascriptversion"   value="1.0" />
            <capability name="javascript"          value="true" />
        </capabilities>
    </browser>

    <gateway id="WebTVbeta" parentID="WebTV">
        <identification>
            <capability name="letters" match="^b" />
        </identification>

        <capture>
        </capture>
        <capabilities>
            <capability name="beta"    value="true" />
        </capabilities>
    </gateway>
</browsers>
```

The advantage of this new method for storing browser capability information is that as new browsers are created or new versions are released, developers simply create or update a `.browser` file to describe the capabilities of that browser.

Accessing Browser Capability Information

Now that you have seen how ASP.NET stores browser capability information, we want to discuss how you can access this information at runtime and program your control to change what it renders based on the browser. To access capability information about the requesting browser, you can use the `Page.Request.Browser` property. This property gives you access to the `System.Web.HttpBrowserCapabilities` class, which provides information about the capabilities of the browser making the current request. The class provides you with a myriad of attributes and properties that describe what the browser can support and render and what it requires. Lists use this information to add capabilities to the TextBox control. Listing 26-27 shows how you can detect browser capabilities to make sure a browser supports JavaScript.

Listing 26-27: Detecting browser capabilities in server-side code

VB

```
Protected Overrides Sub OnPreRender(ByVal e As System.EventArgs)
    If (Page.Request.Browser.EcmaScriptVersion.Major > 0) Then
        Page.ClientScript.RegisterStartupScript( _
            GetType(Page), "ClientCallback", _
            "function ClientCallback() {" & _
                "args=document.getElementById("' & _
                    Me.ClientID & "_i" & "').value;" & _
                Page.ClientScript.GetCallbackEventReference(Me, "args", _
                   "CallbackHandler", Nothing, "ErrorHandler", True) + "}", _
            True)

        Page.ClientScript.RegisterStartupScript(GetType(Page), _
            "ControlFocus", "document.getElementById("' & _
            Me.ClientID & "').focus();", _
            True)
    End If
End Sub
```

C#

```
protected override void OnPreRender(EventArgs e)
{
    if (Page.Request.Browser.EcmaScriptVersion.Major > 0)
    {
        Page.ClientScript.RegisterClientScriptInclude(
            "UtilityFunctions", "JScript.js");
            Page.ClientScript.RegisterStartupScript(
                typeof(Page),
                "ControlFocus","document.getElementById('" +
                    this.ClientID + "_i" + "').focus();",
                true);

            Page.ClientScript.RegisterStartupScript(
                typeof(Page), "ClientCallback",
                "function ClientCallback() {" +
                    "args=document.getElementById('" +
                        this.ClientID + "_i" + "').value;" +
                    Page.ClientScript.GetCallbackEventReference(this, "args",
                        "CallbackHandler", null,"ErrorHandler",true) + "}",
                true);
        }
    }
```

This is a very simple sample, but it gives you an idea of what is possible using the `HttpBrowserCapabilities` class.

Using ViewState

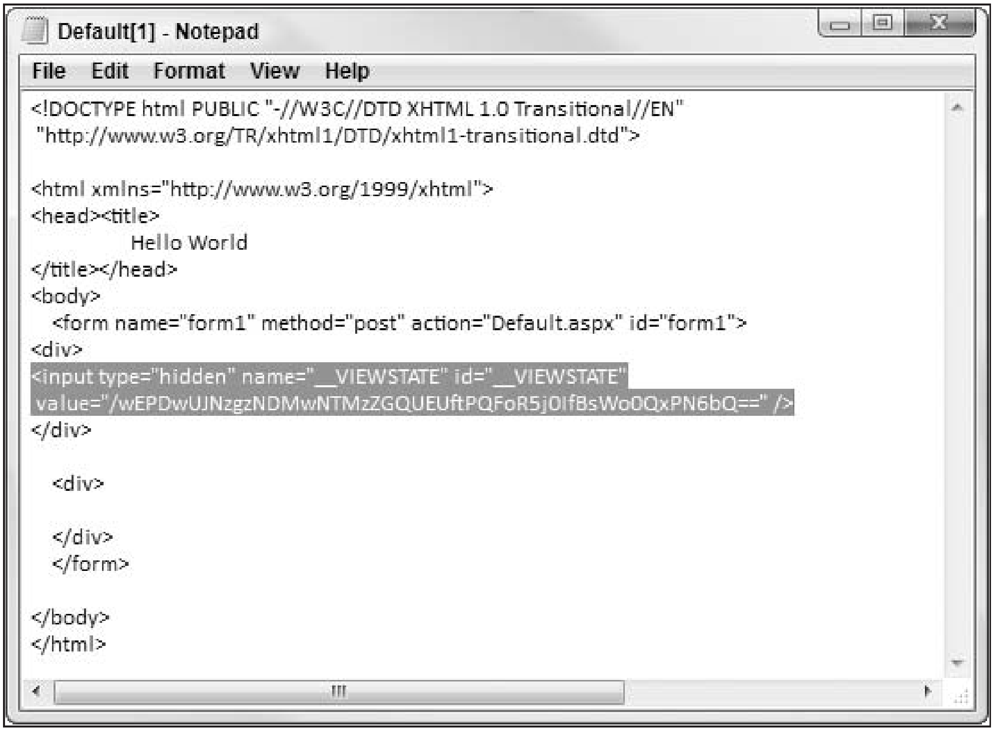
When developing Web applications, remember that they are built on the stateless HTTP protocol. ASP.NET gives you a number of ways to give users the illusion that they are using a stateful application, including Session State and cookies. Additionally, ASP.NET 1.0 introduced a new way of creating the state illusion called ViewState. ViewState enables you to maintain the state of the objects and controls that are part of the Web page through the page's lifecycle by storing the state of the controls in a hidden form field that is rendered as part of the HTML. The state contained in the form field can then be used by the application to reconstitute the page's state when a postback occurs. Figure 26-14 shows how ASP.NET stores ViewState information in a hidden form field.

Notice that the page contains a hidden form field named `_ViewState`. The value of this form field is the ViewState for your Web page. By default, ViewState is enabled in all in-box server controls shipped with ASP.NET. If you write customer server controls, however, you are responsible for ensuring that a control is participating in the use of ViewState by the page.

The ASP.NET ViewState is basically a StateBag that enables you to save and retrieve objects as key/value pairs. As you see in Figure 26-14, these objects are then serialized by ASP.NET and persisted as an encrypted string, which is pushed to the client as a hidden HTML form field. When the page posts back to the server, ASP.NET can use this hidden form field to reconstitute the StateBag, which you can then access as the page is processed on the server.

> *Because the ViewState can sometimes grow to be very large and can therefore affect the overall page size, you might consider an alternate method of storing the ViewState information. You can create your own persistence mechanism by deriving a class from the* `System.Web.UI.PageStatePersister` *class and overriding its* `Load` *and* `Save` *methods.*

As shown in Listing 26-28, by default, the Text property included with the ASP.NET Server Control template is set up to store its value in ViewState.


Default[1] - Notepad
File Edit Format View Help

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
        Hello World
</title></head>
<body>
  <form name="form1" method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
 value="/wEPDwUJNzgzNDMwNTMzZGQUEUftPQFoR5j0IfBsWo0QxPN6bQ==" />
</div>

  <div>

  </div>
  </form>

</body>
</html>
```

Figure 26-14

Listing 26-28: The Text property's use of ViewState

VB

```
Property Text() As String
    Get
        Dim s As String = CStr(ViewState("Text"))
        If s Is Nothing Then
            Return "[" + Me.ID + "]"
        Else
            Return s
        End If
    End Get

    Set(ByVal Value As String)
        ViewState("Text") = Value
    End Set
End Property
```

C#

```
public string Text
{
    get
    {
        String s = (String)ViewState["Text"];
```

```
            return ((s == null) ? "[" + this.ID + "]": s);
        }
        set
        {
            ViewState["Text"] = value;
        }
    }
```

When creating new properties in an ASP.NET server control, you should remember to use this same technique in order to ensure that the values set by the end user in your control will be persisted across page postbacks.

> *Note that the loading of ViewState happens after the* `OnInit` *event has been raised by the page. If your control makes changes to itself or another server control before the event has been raised, the changes are not saved to the ViewState.*

Types and ViewState

As mentioned in the preceding section, the ViewState is basically a generic collection of objects, but not all objects can be added to the ViewState. Only types that can be safely persisted can be used in the ViewState, so objects such as database connections or file handles should not be added to the ViewState.

Additionally, certain data types are optimized for use in the ViewState. When adding data to the ViewState, try to package the data into these types:

- Primitive Types (Int32, Boolean, and so on)
- Arrays of Primitive Types
- ArrayList, HashTable
- Pair, Triplet
- Color, DataTime
- String, IndexedString
- HybridDictionary of these types
- Objects that have a TypeConverter available. Be aware, however, that there is a reduction in performance if you use these types.
- Objects that are serializable (marked with the `Serializable` attribute)

ASP.NET 2.0 introduced new ViewState features that improve performance. For example, the .NET 1.1 ViewState used the LosFormatter to serialize objects, but starting with.NET 2.0, ViewState no longer uses this serializer. Instead, it uses the ObjectStateFormatter, which results in dramatic improvements in the speed with which objects are serialized and deserialized. It also decreases the overall byte size of the resulting serialization. Additionally, beginning with ASP.NET 2.0, ViewState was modified to be written out as bytes rather than strings, thereby saving the cost of converting to a string.

Control State

At times, your control must store critical, usually private information across postbacks. In ASP.NET 1.0, you might have considered using ViewState, but a developer using your control could disable ViewState.

ASP.NET solved this problem by introducing a new kind of ViewState called ControlState. ControlState is essentially a private ViewState for your control only, and it is not affected when ViewState is turned off.

Two new methods, `SaveViewState` and `LoadViewState`, provide access to ControlState; however, the implementation of these methods is left up to you. Listing 26-29 shows how you can use the `LoadControlState` and `SaveViewState` methods.

Listing 26-29: Using ControlState in a server control

VB

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Text
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls

<DefaultProperty("Text")> _
<ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")> _
Public Class ServerControl1
    Inherits WebControl

    Dim s As String
    Protected Overrides Sub OnInit(ByVal e As System.EventArgs)
        Page.RegisterRequiresControlState(Me)
        MyBase.OnInit(e)
    End Sub

    Protected Overrides Sub LoadControlState(ByVal savedState As Object)
        s = CStr(savedState)
    End Sub

    Protected Overrides Function SaveControlState() As Object
        Return CType("FOO", Object)
    End Function

    Protected Overrides Sub Render(ByVal output As System.Web.UI.HtmlTextWriter)
        output.Write("Control State: " & s)
    End Sub

End Class
```

C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
```

```
namespace ServerControl1
{
    [DefaultProperty("Text")]
    [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")]
    public class ServerControl1 : WebControl
    {
        string s;
        protected override void OnInit(EventArgs e)
        {
            Page.RegisterRequiresControlState(this);
            base.OnInit(e);
        }

        protected override void LoadControlState(object savedState)
        {
            s = (string)savedState;
        }

        protected override object SaveControlState()
        {
            return (object)"FOO";
        }

        protected override void Render(HtmlTextWriter output)
        {
            output.Write("Control State: " + s);
        }
    }
}
```

Controls intending to use ControlState must call the `Page.RegisterRequiresControlState` method before attempting to save control state data. Additionally, the `RegisterRequiresControlState` method must be called for each page load because the value is not retained through page postbacks.

Raising PostBack Events

As you have seen in this chapter, ASP.NET provides a very powerful set of tools you can use to develop server controls and emit them to a client browser. But this is still one-way communication because the server only pushes data to the client. It would be useful if the server control could send data back to the server. The process of sending data back to the server is generally known as a *page postback*. You experience a page postback any time you click a form button or link that causes the page to make a new request to the Web server.

ASP.NET provides a rich framework for handling postbacks from ASP.NET Web pages. Additionally, ASP.NET attempts to give you a development model that mimics the standard Windows Forms event model. It enables you to use controls that, even though they are rendered in the client browser, can raise events in server-side code. It also provides an easy mechanism for plugging the server control into that framework, allowing you to create controls that can cause a page postback. Figure 26-15 shows the ASP.NET postback framework.

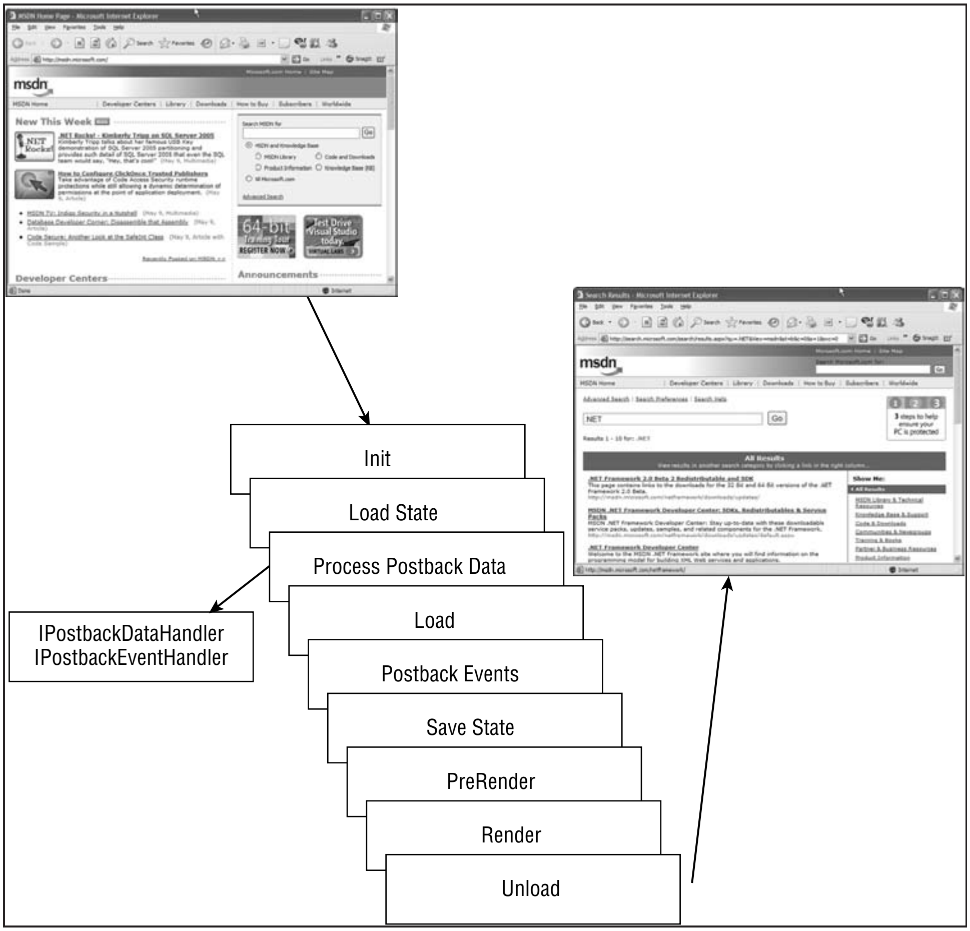

Init
Load State
Process Postback Data
IPostbackDataHandler
IPostbackEventHandler
Load
Postback Events
Save State
PreRender
Render
Unload

Figure 26-15

In order to initiate a postback, ASP.NET uses client-side scripting. You can add the proper script to your control by using the `GetPostBackEventReference` method and emitting the results to the client during the controls render method. Listing 26-30 shows how you can add that to a new server control that emits an HTML button.

Listing 26-30: Adding PostBack capabilities to a server control

VB

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Text
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls
```

```
<DefaultProperty("Text")> _
<ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")> _
Public Class ServerControl1
    Inherits WebControl

    Protected Overrides Sub RenderContents(ByVal output As HtmlTextWriter)
        Dim p As New PostBackOptions(Me)

        output.AddAttribute(HtmlTextWriterAttribute.Onclick, _
            Page.ClientScript.GetPostBackEventReference(p))

        output.AddAttribute(HtmlTextWriterAttribute.Value, "My Button")
        output.AddAttribute(HtmlTextWriterAttribute.Id, Me.ClientID & "_i")
        output.AddAttribute(HtmlTextWriterAttribute.Name, Me.ClientID & "_i")
        output.RenderBeginTag(HtmlTextWriterTag.Button)
        output.RenderEndTag()
    End Sub
End Class
```

C#

```
Using System;
Using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ServerControl1
{
    [DefaultProperty("Text")]
    [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")]
    public class ServerControl1 : WebControl
    {
        protected override void RenderContents(HtmlTextWriter output)
        {
            PostBackOptions p = new PostBackOptions(this);

            output.AddAttribute(HtmlTextWriterAttribute.Onclick,
                Page.ClientScript.GetPostBackEventReference(p));
            output.AddAttribute(HtmlTextWriterAttribute.Value, "My Button");
            output.AddAttribute(HtmlTextWriterAttribute.Id, this.ClientID + "_i");
            output.AddAttribute(HtmlTextWriterAttribute.Name,
                this.ClientID + "_i");
            output.RenderBeginTag(HtmlTextWriterTag.Button);
            output.RenderEndTag();
        }
    }
}
```

As you can see, this code adds the postback event reference to the client-side OnClick event, but you are not limited to that. You can add the postback JavaScript to any client-side event. You could even add the code to a client-side function if you want to include some logic code.

Now that you can create a postback, you may want to add events to your control that execute during the page postback. To raise server-side events from a client-side object, you implement the System.Web.IPostBackEventHandler interface. Listing 26-31 shows how to do this for a button control. You also create a server-side Click event you can handle when the page posts back.

Listing 26-31: Handling postback events in a server control

VB

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Text
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls

<DefaultProperty("Text")> _
<ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")>  _
Public Class ServerControl1
    Inherits System.Web.UI.WebControls.WebControl
    Implements IPostBackEventHandler

    '. . . Code removed for clarity . . .

    Public Event Click()
    Public Sub OnClick(ByVal args As EventArgs)
        RaiseEvent Click()
    End Sub

    Public Sub RaisePostBackEvent(ByVal eventArgument As String) _
      Implements System.Web.UI.IPostBackEventHandler.RaisePostBackEvent
        OnClick(EventArgs.Empty)
    End Sub
End Class
```

C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ServerControl1
{
    [DefaultProperty("Text")]
    [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")]
    public class ServerControl1 : WebControl, IPostBackEventHandler
    {

        //. . . Code removed for clarity . . .

        #region IPostBackEventHandler Members
        public event EventHandler Click;

        public virtual void OnClick(EventArgs e)
        {
            if (Click != null)
            {
                Click(this,e);
            }
        }

        public void RaisePostBackEvent(string eventArgument)
        {
            OnClick(EventArgs.Empty);
        }

        #endregion
    }
}
```

Now, when the user clicks the button and the page posts back, the server-side Click event fires, allowing you to add server-side handling code to the event.

Handling PostBack Data

Now that you have learned how to store data in ViewState and add postback capabilities to a control, look at how you can enable the control to interact with data the user enters into one of its form fields. When a page is posted back to the server by ASP.NET, all the form data is also posted to the server. If the control can interact with data that is passed with a page, you can store the information in ViewState and complete the illusion of a stateful application.

To interact with postback data, your control must be able to access the data. To do this, it implements the System.Web.IPostBackDataHandler interface. This interface allows your control to examine the form data that is passed back to the server during the postback.

The IPostBackDataHandler interface requires that you implement two methods: `LoadPostData` and `RaisePostBackDataChangedEvent`. The `LoadPostData` method is called for all server controls on the page that have postback data. If a control does not have any postback data, the method is not called; however, you can explicitly ask for the method to be called by using the `RegisterRequiresPostBack` method.

Listing 26-32 shows how you implement the IPostBackDataHandler interface method in a text box.

Listing 26-32: Accessing Postback data in a server control

VB

```
Imports System
Imports Sustem.Collections.Generic
Imports System.ComponentModel
Imports System.Text
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls
<DefaultProperty("Text")> _
```

```
<ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")> _
Public Class ServerControl1
    Inherits SystemWebControl
    Implements IPostBackEventHandler, IPostBackDataHandler

    '. . . Code removed for clarity . . .

    Public Function LoadPostData(ByVal postDataKey As String, _
        ByVal postCollection As _
            System.Collections.Specialized.NameValueCollection) _
        As Boolean Implements System.Web.UI.IPostBackDataHandler.LoadPostData

        Me.Text = postCollection(postDataKey)
        Return False
    End Function

    Public Sub RaisePostDataChangedEvent() _
        Implements System.Web.UI.IPostBackDataHandler.RaisePostDataChangedEvent

    End Sub

End Class
```

C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ServerControl1
{
    [DefaultProperty("Text")]
    [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")]
    public class ServerControl1 : WebControl,
        IPostBackEventHandler, IPostBackDataHandler
    {

        //. . . Code removed for clarity . . .

        public bool LoadPostData(string postDataKey,
             System.Collections.Specialized.NameValueCollection postCollection)
        {
            this.Text = postCollection[postDataKey];
            return false;
        }

        public void RaisePostDataChangedEvent()
        {
        }
    }
}
```

As you can see, the `LoadPostData` method passes any form data submitted to the method as a name value collection that the control can access. The `postDataKey` parameter allows the control to access the postback data item specific to it. You use these parameters to save text to the `Text` property of the TextBox control. If you remember the earlier ViewState example, the `Text` property saves the new value to ViewState; when the page renders, the TextBox value automatically repopulates.

In addition to the input parameters, the `LoadPostData` method also returns a Boolean value. This value indicates whether the `RaisePostBackDataChangedEvent` method is also called after the `LoadPostData` method completes execution. In the sample, it returns `false` because no events exist, but if you create a `TextChanged` event to indicate the Textbox text has changed, you raise that event in the `RaisePostDataChangedEvent` method.

Composite Controls

So far, in looking at Server controls, you have concentrated on emitting a single HTML control; but this can be fairly limiting. Creating extremely powerful controls often requires that you nest several HTML elements together. ASP.NET allows you to easily create controls that serve as a container for other controls. These types of controls are called *composite controls*.

To demonstrate how easy creating a composite control can be, try to change an existing control into a composite control. Listing 26-33 shows how you can do this.

Listing 26-33: Creating a composite control

VB

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Text
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls

<DefaultProperty("Text")> _
<ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")> _
Public Class ServerControl1
    Inherits System.Web.UI.WebControls.CompositeControl

    Protected textbox As TextBox

    Protected Overrides Sub CreateChildControls()
        Me.Controls.Add(textbox)
    End Sub

End Class
```

C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ServerControl1
{
    [DefaultProperty("Text")]
    [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")]
    public class ServerControl1 : CompositeControl
    {
        protected TextBox textbox = new TextBox();

        protected override void CreateChildControls()
        {
            this.Controls.Add(textbox);
        }
    }
}
```

A number of things in this listing are important. First, notice that the control class is now inheriting from `CompositeControl`, rather than `WebControl`. Deriving from `CompositeControl` gives you a few extra features specific to this type of control.

Second, notice that no `Render` method appears in this code. Instead, you simply create an instance of another type of server control and add that to the `Controls` collection in the `CreateChildControls` method. When you run this sample, you see that it renders a text box just like the last control did. In fact, the HTML that it renders is almost identical.

Exposing Child Control Properties

When you drop a composite control (such as the text box from the last sample) onto the design surface, notice that even though you are using a powerful ASP.NET TextBox control within the control, none of that control's properties are exposed to you in the Properties Explorer. In order to expose child control properties through the parent container, you must create corresponding properties in the parent control. For example, if you want to expose the ASP.NET text box `Text` property through the parent control, you create a `Text` property. Listing 26-34 shows how to do this.

Listing 26-34: Exposing control properties in a composite control

VB

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Text
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls

<DefaultProperty("Text")> _
<ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")> _
Public Class ServerControl1
    Inherits System.Web.UI.WebControls.CompositeControl

    Protected textbox As TextBox

    Public Property Text() As String
```

```
        Get
            EnsureChildControls()
            Return textbox.Text
        End Get
        Set(ByVal value As String)
            EnsureChildControls()
            textbox.Text = value
        End Set
    End Property

    Protected Overrides Sub CreateChildControls()
        Me.Controls.Add(textbox)
        Me.ChildControlsCreated=True
    End Sub

End Class
```

C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ServerControl1
{
    [DefaultProperty("Text")]
    [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")]
    public class ServerControl1 : CompositeControl
    {

        protected TextBox textbox = new TextBox();
        public string Text
        {
            get
            {
                EnsureChildControls();
                return textbox.Text;
            }
            set
            {
                EnsureChildControls();
                textbox.Text = value;
            }
        }

        protected override void CreateChildControls()
        {
            this.Controls.Add(textbox);
            this.ChildControlsCreated=true;
        }
    }
}
```

Notice that you use this property simply to populate the underlying control's properties. Also notice that before you access the underlying control's properties, you always call the `EnsureChildControls` method. This method ensures that children of the container control have actually been initialized before you attempt to access them.

Templated Controls

In addition to composite controls, you can also create templated controls. *Templated controls* allow the user to specify a portion of the HTML that is used to render the control, and to nest other controls inside of a container control. You might be familiar with the Repeater or DataList control. These are both templated controls that let you specify how you want the bound data to be displayed when the page renders.

To demonstrate a templated control, the following code gives you a simple example of displaying a message from a user on a Web page. Because the control is a templated control, the developer has complete control over how the message is displayed.

To get started, create the Message server control that will be used as the template inside of a container control. Listing 26-35 shows the class which simply extends the existing Panel control by adding two additional properties, Name and Text, and a new constructor.

Listing 26-35: Creating the templated control's inner control class

VB

```
Public Class Message
    Inherits System.Web.UI.WebControls.Panel
    Implements System.Web.UI.INamingContainer

    Private _name As String
    Private _text As String

    Public Sub New(ByVal name As String, ByVal text As String)
        _text = text
        _name = name
    End Sub

    Public ReadOnly Property Name() As String
        Get
            Return _name
        End Get
    End Property

    Public ReadOnly Property Text() As String
        Get
            Return _text
        End Get
    End Property
End Class
```

C#

```
using System;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ServerControl1
{
    public class Message : Panel, INamingContainer
    {
        private string _name;
        private string _text;

        public Message(string name, string text)
        {
            _text = text;
            _name = name;
        }

        public string Name
        {
            get { return _name;}
        }

        public string Text
        {
            get { return _text;}
        }
    }
}
```

As you will see in a moment, you can access the public properties exposed by the `Message` class in order to insert dynamic content into the template. You will also see how you can display the values of the `Name` and `Text` properties as part of the rendered template control.

Next, as shown in Listing 26-36, create a new server control which will be the container for the `Message` control. This server control is responsible for rendering any template controls nested in it.

Listing 26-36: Creating the template control container class

VB

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Text
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls

<DefaultProperty("Text")> _
<ToolboxData("<{0}:TemplatedControl runat=server></{0}:TemplatedControl>")> _
Public Class TemplatedControl
    Inherits System.Web.UI.WebControls.WebControl

    Private _name As String
    Private _text As String
Private _message As Message
Private _messageTemplate As ITemplate

<Browsable(False)> Public ReadOnly Property Message() As Message
    Get
        Return _message
    End Get
End Property

<PersistenceMode(PersistenceMode.InnerProperty), _
    TemplateContainer(GetType(Message))> _
Public Property MessageTemplate() As ITemplate
    Get
        Return _messageTemplate
    End Get
    Set(ByVal value As ITemplate)
        _messageTemplate = value
    End Set
End Property

<Bindable(True), DefaultValue("")> Public Property Name() As String
    Get
        Return _name
    End Get
    Set(ByVal value As String)
        _name = value
    End Set
End Property

<Bindable(True), DefaultValue("")> Public Property Text() As String
    Get
        Return _text
    End Get
    Set(ByVal value As String)
        _text = value
    End Set
End Property

Public Overrides Sub DataBind()
    CreateChildControls()
    ChildControlsCreated = True

    MyBase.DataBind()
End Sub

Protected Overrides Sub CreateChildControls()

    Me.Controls.Clear()

    _message = New Message(Name, Text)

    If Me.MessageTemplate Is Nothing Then
        Me.MessageTemplate = New DefaultMessageTemplate()
    End If
```

```
            Me.MessageTemplate.InstantiateIn(_message)
            Controls.Add(_message)
        End Sub

        Protected Overrides Sub RenderContents( _
            ByVal writer As System.Web.UI.HtmlTextWriter)

            EnsureChildControls()
            ChildControlsCreated = True

            MyBase.RenderContents(writer)
        End Sub
    End Class
```

C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ServerControl1
{
   [DefaultProperty("Text")]
   [ToolboxData("<{0}:TemplatedControl runat=server></{0}: TemplatedControl >")]
    public class TemplatedControl : WebControl
    {
        private string _name;
        private string _text;

        private Message _message;
        private ITemplate _messageTemplate;

        [Browsable(false)]
        public Message Message
        {
            get
            {
                return _message;
            }
        }

        [PersistenceMode(PersistenceMode.InnerProperty)]
        [TemplateContainer(typeof(Message))]
        public virtual ITemplate MessageTemplate
        {
            get { return _messageTemplate;}
            set { _messageTemplate = value;}
        }
        [Bindable(true)]
        [DefaultValue("")]
        public string Name
        {
            get { return _name;}
            set { _name = value;}
        }

        [Bindable(true)]
        [DefaultValue("")]
        public string Text
        {
            get { return _text;}
            set { _text = value;}
        }

        public override void DataBind()
        {
            CreateChildControls();
            ChildControlsCreated = true;

            base.DataBind();
        }

        protected override void CreateChildControls()
        {
            this.Controls.Clear();

            _message = new Message(Name,Text);

            if (this.MessageTemplate == null)
            {
                this.MessageTemplate = new DefaultMessageTemplate();
            }
            this.MessageTemplate.InstantiateIn(_message);
            Controls.Add(_message);
        }

        protected override void RenderContents(HtmlTextWriter writer)
        {
            EnsureChildControls();
            ChildControlsCreated = true;

            base.RenderContents(writer);
        }
    }
}
```

To start to dissect this sample, first notice the `MessageTemplate` property. This property allows Visual Studio to understand that the control can contain a template, and allows it to display the IntelliSense for that template. The property has been marked with the `PersistanceMode` attribute indicating that the template control should be persisted as an inner property within the control's tag in the ASPX page. Additionally, the property is marked with the `TemplateContainer` attribute, which helps ASP.NET figure out what type of template control this property represents. In this case, it's the Message template control you created earlier.

The container control exposes two public properties, Name and Text. These properties are used to populate the `Name` and `Text` properties of the Message control since that class does not allow developers to set the properties directly.

Finally, the `CreateChildControls` method, called by the `DataBind` method, does most of the heavy lifting in this control. It creates a new `Message` object, passing the values of `Name` and `Text` as constructor values. Once the `CreateChildControls` method completes, the base DataBind operation comtinues to execute. This is important because that is where the evaluation of the Name and Text properties occurs, which allows you to insert these properties values into the template control.

After the control and template are created, you can drop them onto a test Web page. Listing 26-37 shows how the control can be used to customize the display of the data.

Listing 26-37: Adding a templated control to a Web page

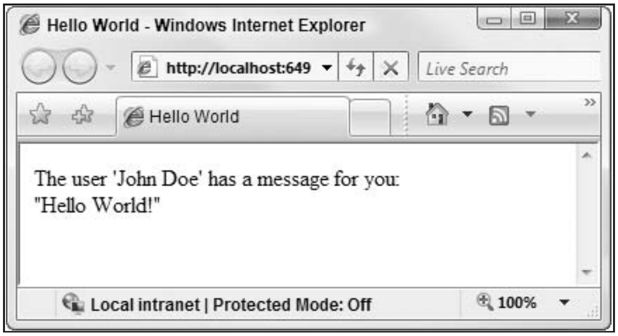
VB

```
<%@ Page Language="VB" %>

<%@ Register Assembly="WebControlLibrary1" Namespace="WebControlLibrary1"
    TagPrefix="cc1" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Me.TemplatedControl1.DataBind()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Templated Web Controls</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <cc1:TemplatedControl Name="John Doe" Text="Hello World!"
            ID="TemplatedControl1" runat="server">
            <MessageTemplate>The user '<%# Container.Name %>'
                has a message for you: <br />"<%#Container.Text%>"
            </MessageTemplate>
        </cc1:TemplatedControl>
    </div>
    </form>
</body>
</html>
```

C#

```
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        this.TemplatedControl1.DataBind();
    }
</script>
```

As you can see in the listing, the `<cc1:TemplatedControl>` control contains a `MessageTemplate` within it, which has been customized to display the `Name` and `Text` values. Figure 26-16 shows this page after it has been rendered in the browser.


Hello World - Windows Internet Explorer
http://localhost:649
Live Search
Hello World
Local intranet | Protected Mode: Off
100%

The user 'John Doe' has a message for you:
"Hello World!"

Figure 26-16

One item to consider when creating templated controls is what happens if the developer does not include a template control inside of the container control. In the previous example, if you removed the MessageTemplate control from the TemplateContainer, a NullReferenceException would occur when you tried to run your Web page because the container control's MessageTemplate property would return a null value. In order to prevent this, you can include a default template class as part of the container control. An example of a default template is shown in Listing 26-38.

Listing 26-38: Creating the templated control's default template class

VB

```
Friend Class DefaultMessageTemplate
    Implements ITemplate

    Public Sub InstantiateIn(ByVal container As System.Web.UI.Control) _
            Implements System.Web.UI.ITemplate.InstantiateIn

        Dim l As New Literal()
        l.Text="No MessageTemplate was included."
        container.Controls.Add(l)
    End Sub
End Class
```

C#

```
internal sealed class DefaultMessageTemplate : ITemplate
{
    public void InstantiateIn(Control container)
    {
        Literal l = new Literal();
        l.Text="No MessageTemplate was included.";
        container.Controls.Add(l);
    }
}
```

Notice that the DefaultMessageTemplate implements the ITemplate interface. This interface requires that the `InstantiateIn` method be implemented, which we use to provide the default template content.

To include the default template, simply add the class to the `TemplatedControl` class. You will also need to modify the `CreateChildControls` method to detect the null MessageTemplate and instead create an instance of and use the default template.

VB

```
If template = Nothing Then
    template = New DefaultMessageTemplate()
End If
```

C#

```
if (template == null)
{
    template = new DefaultMessageTemplate();
}
```

Creating Control Design-Time Experiences

So far in this chapter, you concentrated primarily on what gets rendered to the client's browser, but the browser is not the only consumer of server controls. Visual Studio and the developer using a server control are also consumers, and you need to consider their experiences when using your control.

> *Note that beginning with Visual Studio 2008, the Web Page Design Surface used to provide Web page designers with a WYSIWYG design experience has been completely rewritten. The design-surface, which in prior versions was derived from the core Internet Explorer rendering engine has been replaced by a completely independent and new rendering engine. This is good news for Web page developers because they are no longer subject to the quirks of IE rendering. If you have existing controls you should be sure to test them thoroughly on the new design-surface to ensure compatibility. From a control design perspective, all of the previous functionality has been retained. Therefore, any controls you have written to take advantage of design-time tools such as SmartTags or Designer regions should function normally on the new design surface.*

ASP.NET offers numerous improvements in the design-time experience you give to developers using your control. Some of these improvements require no additional coding, such as the WYSIWYG rendering of user controls and basic server controls; but for more complex scenarios, ASP.NET includes a number of tools that give the developer an outstanding design-time experience.

When you write server controls, a priority should be to give the developer a design-time experience that closely replicates the runtime experience. This means altering the appearance of the control on the design surface in response to changes in control properties and the introduction of other server controls onto the design surface. Three main components are involved in creating the design-time behaviors of a server control:

- ❑ `Type Converters`
- ❑ `Designers`
- ❑ `UI Type Editors`

Because a chapter can be written for each one of these topics, in this section I attempt to give you only an overview of each, how they tie into a control's design-time behavior, and some simple examples of their use.

Type Converters

`TypeConverter` is a class that allows you to perform conversions between one type and another. Visual Studio uses type converters at design time to convert object property values to String types so that they can be displayed on the Property Browser, and it returns them to their original types when the developer changes the property.

ASP.NET includes a wide variety of type converters you can use when creating your control's design-time behavior. These range from converters that allow you to convert most number types, to converters that let you convert Fonts, Colors, DataTimes, and Guids. The easiest way to see what type converters are available to you in the .NET Framework is to search for types in the framework that derive from the `TypeConverter` class using the MSDN Library help.

After you have found a type converter that you want to use on a control property, mark the property with a `TypeConverter` attribute, as shown in Listing 26-39.

Listing 26-39: Applying the TypeConverter attribute to a property

VB

```
<Bindable(True)> _
<Category("Appearance")> _
<DefaultValue("")> _
<TypeConverter(GetType(GuidConverter))> _
Property BookId() As System.Guid
        Get
            Return _bookid
        End Get

        Set(ByVal Value As System.Guid)
            _bookid = Value
        End Set
End Property
```

C#

```
[Bindable(true)]
[Category("Appearance")]
[DefaultValue("")]
[TypeConverter(typeof(GuidConverter))]
public Guid BookId
{
    get
    {
        return _bookid;
    }

    set
    {
        _bookid = value;
    }
}
```

In this example, a property is exposed that accepts and returns an object of type Guid. The Property Browser cannot natively display a Guid object, so you convert the value to a string so that it can be displayed properly in the property browser. Marking the property with the `TypeConverter` attribute and, in this case, specifying the GuidConverter as the type converter you want to use, allows complex objects like a Guid to display properly in the Property Browser.

Custom Type Converters

It is also possible to create your own custom type converters if none of the in-box converters fit into your scenario. Type converters derive from the `System.ComponentModel.TypeConverter` class. Listing 26-40 shows a custom type converter that converts a custom object called `Name` to and from a string.

Listing 26-40: Creating a custom type converter

VB

```
Imports System
Imports System.ComponentModel
Imports System.Globalization

Public Class Name

    Private _first As String
    Private _last As String

    Public Sub New(ByVal first As String, ByVal last As String)
        _first = first
        _last = last
    End Sub

    Public Property First() As String
        Get
            Return _first
        End Get
        Set(ByVal value As String)
            _first = value
        End Set
    End Property

    Public Property Last() As String
        Get
            Return _last
        End Get
        Set(ByVal value As String)
            _last = value
        End Set
    End Property
End Class

Public Class NameConverter
    Inherits TypeConverter

    Public Overrides Function CanConvertFrom(ByVal context As _
       ITypeDescriptorContext, ByVal sourceType As Type) As Boolean

        If (sourceType Is GetType(String)) Then
            Return True
        End If

        Return MyBase.CanConvertFrom(context, sourceType)
    End Function
    Public Overrides Function ConvertFrom( _
            ByVal context As ITypeDescriptorContext, _
            ByVal culture As CultureInfo, ByVal value As Object) As Object
        If (value Is GetType(String)) Then
            Dim v As String() = (CStr(value).Split(New [Char]() {" "c}))
            Return New Name(v(0), v(1))
        End If
        Return MyBase.ConvertFrom(context, culture, value)
    End Function

    Public Overrides Function ConvertTo( _
            ByVal context As ITypeDescriptorContext, _
            ByVal culture As CultureInfo, ByVal value As Object, _
            ByVal destinationType As Type) As Object
        If (destinationType Is GetType(String)) Then
            Return (CType(value, Name).First + " " + (CType(value, Name).Last))
        End If
        Return MyBase.ConvertTo(context, culture, value, destinationType)
    End Function
End Class
```

C#

```
using System;
using System.ComponentModel;
using System.Globalization;

public class Name
{
    private string _first;
    private string _last;

    public Name(string first, string last)
    {
        _first=first;
        _last=last;
    }

    public string First
    {
        get{ return _first;}
        set { _first = value;}
    }
    public string Last
    {
        get { return _last;}
        set { _last = value;}
    }
}

public class NameConverter : TypeConverter
{

   public override bool CanConvertFrom(ITypeDescriptorContext context,
      Type sourceType) {
            if (sourceType == typeof(string)) {
               return true;
            }
            return base.CanConvertFrom(context, sourceType);
         }

         public override object ConvertFrom(ITypeDescriptorContext context,
            CultureInfo culture, object value) {
            if (value is string) {
               string[] v = ((string)value).Split(new char[] {' '});
               return new Name(v[0],v[1]);
            }
            return base.ConvertFrom(context, culture, value);
         }

         public override object ConvertTo(ITypeDescriptorContext context,
            CultureInfo culture, object value, Type destinationType) {
            if (destinationType == typeof(string)) {
               return ((Name)value).First + " " + ((Name)value).Last;
            }
            return base.ConvertTo(context, culture, value, destinationType);
         }
      }
```

The `NameConverter` class overrides three methods, `CanConvertFrom`, `ConvertFrom`, and `ConvertTo`. The `CanConvertFrom` method allows you to control what types the converter can convert from. The `ConvertFrom` method converts the string representation back into a `Name` object, and `ConvertTo` converts the `Name` object into a string representation.

After you have built your type converter, you can use it to mark properties in your control with the `TypeConverter` attribute, as you saw in Listing 26-35.

Control Designers

Controls that live on the Visual Studio design surface depend on *control designers* to create the design-time experience for the end user. Control designers, for both WinForms and ASP.NET, are classes that derive from the `System.ComponentModel.Design.ComponentDesigner` class. .NET provides an abstracted base class specifically for creating ASP.NET control designers called the `System.Web.UI.Design.ControlDesigner`. In order to access these classes you will need to add a reference to the System.Design.dll assembly to your project.

.NET includes a number of in-box control designer classes that you can use when creating a custom control; but as you develop server controls, you see that .NET automatically applies a default designer. The designer it applies is based on the type of control you are creating. For instance, when you created your first TextBox control, Visual Studio used the `ControlDesigner` class to achieve the WYSIWYG design-time rendering of the text box. If you develop a server control derived from the `ControlContainer` class, .NET automatically use the `ControlContainerDesigner` class as the designer.

You can also explicitly specify the designer you want to use to render your control at design time using the `Designer` attribute on your control's class, as shown in Listing 26-41.

Listing 26-41: Adding a Designer attribute to a control class

VB

```
<DefaultProperty("Text")> _
<ToolboxData("<{0}:WebCustomControl1 runat=server></{0}:WebCustomControl1>")> _
<Designer(GetType(System.Web.UI.Design.ControlDesigner))> _
Public Class WebCustomControl1
    Inherits System.Web.UI.WebControls.WebControl
```

C#

```
[DefaultProperty("Text")]
[ToolboxData("<{0}:WebCustomControl1 runat=server></{0}:WebCustomControl1>")]
[Designer(typeof(System.Web.UI.Design.ControlDesigner))]
public class WebCustomControl1 : WebControl
```

Notice that the `Designer` attribute has been added to the `WebCustomControl1` class. You have specified that the control should use the `ControlDesigner` class as its designer. Other in-box designers you could have specified are

- ❑ `CompositeControlDesigner`
- ❑ `TemplatedControlDesigner`
- ❑ `DataSourceDesigner`

Each designer provides a specific design-time behavior for the control, and you can select one that is appropriate for the type of control you are creating.

Design-Time Regions

As you saw earlier, ASP.NET allows you to create server controls that consist of other server controls and text. In ASP.NET 1.0, a server control developer could use the `ReadWriteControlDesigner` class to enable the user of the server control to enter text or drop other server controls into a custom server control at design time. An example of this is the ASP.NET Panel control, which enables developers to add content to the panel at design time.

Since ASP.NET 2.0, however, creating a control with this functionality has changed. The `ReadWriteControlDesigner` class was marked as obsolete, and a new and improved way was included to allow the developer to create server controls that have design-time editable portions. The new technique, called *designer regions*, is an improvement over the `ReadWriteControlDesigner` in several ways. First, unlike the `ReadWriteControlDesigner` class, which allowed only a single editable area, designer regions enable you to create multiple, independent regions defined within a single control. Second, designer classes can now respond to events raised by a design region. This might be the designer drawing a control on the design surface or the user clicking an area of the control or entering or exiting a template edit mode.

To show how you can use designer regions, create a container control to which you can apply a custom control designer (as shown in Listing 26-42).

Listing 26-42: Creating a composite control with designer regions

VB

```
<Designer(GetType(MultiRegionControlDesigner))> _
<ToolboxData("<{0}:MultiRegionControl runat=server width=100%>" & _
    "</{0}:MultiRegionControl>")> _
Public Class MultiRegionControl
    Inherits CompositeControl

    ' Define the templates that represent 2 views on the control
    Private _view1 As ITemplate
    Private _view2 As ITemplate

    ' These properties are inner properties
    <PersistenceMode(PersistenceMode.InnerProperty), DefaultValue("")> _
    Public Overridable Property View1() As ITemplate
        Get
            Return _view1
        End Get
        Set(ByVal value As ITemplate)
            _view1 = value
        End Set
    End Property

    <PersistenceMode(PersistenceMode.InnerProperty), DefaultValue("")> _
    Public Overridable Property View2() As ITemplate
        Get
            Return _view2
        End Get
        Set(ByVal value As ITemplate)
            _view2 = value
        End Set
    End Property

    ' The current view on the control; 0= view1, 1=view2, 2=all views
    Private _currentView As Int32 = 0
    Public Property CurrentView() As Int32
        Get
            Return _currentView
        End Get
        Set(ByVal value As Int32)
            _currentView = value
        End Set
    End Property

    Protected Overrides Sub CreateChildControls()
        MyBase.CreateChildControls()

        Controls.Clear()

        Dim template As ITemplate = View1
        If (_currentView = 1) Then
            template = View2
        End If

        Dim p As New Panel()
        Controls.Add(p)

        If (Not template Is Nothing) Then
            template.InstantiateIn(p)
        End If

    End Sub

End Class
```

C#

```
[Designer(typeof(MultiRegionControlDesigner))]
[ToolboxData("<{0}:MultiRegionControl runat=\"server\" width=\"100%\">" +
    "</{0}:MultiRegionControl>")]
public class MultiRegionControl : CompositeControl {

    // Define the templates that represent 2 views on the control
    private ITemplate _view1;
    private ITemplate _view2;

    // These properties are inner properties
    [PersistenceMode(PersistenceMode.InnerProperty), DefaultValue(null)]
    public virtual ITemplate View1 {
        get { return _view1;}
        set { _view1 = value;}
    }

    [PersistenceMode(PersistenceMode.InnerProperty), DefaultValue(null)]
    public virtual ITemplate View2 {
        get { return _view2;}
        set { _view2 = value;}
    }

    // The current view on the control; 0= view1, 1=view2, 2=all views
    private int _currentView = 0;
    public int CurrentView {
        get { return _currentView;}
        set { _currentView = value;}
    }

    protected override void CreateChildControls()
    {
        Controls.Clear();
        ITemplate template = View1;
        if (_currentView == 1)
            template = View2;

        Panel p = new Panel();
        Controls.Add(p);
        if (template != null)
            template.InstantiateIn(p);
    }
}
```

The container control creates two `ITemplate` objects, which serve as the controls to display. The `ITemplate` objects are the control containers for this server control, allowing you to drop other server controls or text into this control. The control also uses the `Designer` attribute to indicate to Visual Studio that it should use the `MultiRegionControlDesigner` class when displaying this control on the designer surface.

Now you create the control designer that defines the regions for the control. Listing 26-43 shows the designer class.

Listing 26-43: A custom designer class used to define designer regions

VB

```
Public Class MultiRegionControlDesigner
    Inherits System.Web.UI.Design.WebControls.CompositeControlDesigner

    Protected _currentView As Int32 = 0
    Private myControl As MultiRegionControl

    Public Overrides Sub Initialize(ByVal component As IComponent)
        MyBase.Initialize(component)
        myControl = CType(component, MultiRegionControl)
    End Sub

    Public Overrides ReadOnly Property AllowResize() As Boolean
        Get
            Return True
        End Get
    End Property

    Protected Overrides Sub OnClick(ByVal e As DesignerRegionMouseEventArgs)

        If (e.Region Is Nothing) Then
            Return
        End If

        If ((e.Region.Name = "Header0") And (Not _currentView = 0)) Then
            _currentView = 0
            UpdateDesignTimeHtml()
        End If

        If ((e.Region.Name = "Header1") And (Not _currentView = 1)) Then

            _currentView = 1
            UpdateDesignTimeHtml()
        End If
    End Sub

    Public Overrides Function GetDesignTimeHtml( _
            ByVal regions As DesignerRegionCollection) As String
        BuildRegions(regions)
        Return BuildDesignTimeHtml()
    End Function

    Protected Overridable Sub BuildRegions( _
            ByVal regions As DesignerRegionCollection)

        regions.Add(New DesignerRegion(Me, "Header0"))
        regions.Add(New DesignerRegion(Me, "Header1"))

        ' If the current view is for all, we need another editable region
        Dim edr0 As New EditableDesignerRegion(Me, "Content" & _currentView, False)
        edr0.Description = "Add stuff in here if you dare:"
        regions.Add(edr0)

        ' Set the highlight, depending upon the selected region
        If ((_currentView = 0) Or (_currentView = 1)) Then
            regions(_currentView).Highlight = True
        End If
    End Sub

    Protected Overridable Function BuildDesignTimeHtml() As String

        Dim sb As New StringBuilder()
        sb.Append(BuildBeginDesignTimeHtml())
        sb.Append(BuildContentDesignTimeHtml())
        sb.Append(BuildEndDesignTimeHtml())

        Return sb.ToString()
    End Function

    Protected Overridable Function BuildBeginDesignTimeHtml() As String
        ' Create the table layout
        Dim sb As New StringBuilder()
        sb.Append("<table ")

        ' Styles that we'll use to render for the design-surface
        sb.Append("height='" & myControl.Height.ToString() & "' width='" & _
            myControl.Width.ToString() & "'>")

        ' Generate the title or caption bar
        sb.Append("<tr height='25px' align='center' " & _
            "style='font-family:tahoma;font-size:10pt;font-weight:bold;'>" & _
            "<td style='width:50%' " & _
            DesignerRegion.DesignerRegionAttributeName & "='0'>")
        sb.Append("Page-View 1</td>")
        sb.Append("<td style='width:50%' " & _
            DesignerRegion.DesignerRegionAttributeName & "='1'>")
        sb.Append("Page-View 2</td></tr>")

        Return sb.ToString()
    End Function

    Protected Overridable Function BuildEndDesignTimeHtml() As String
```

```
        Return ("</table>")
    End Function

    Protected Overridable Function BuildContentDesignTimeHtml() As String

        Dim sb As New StringBuilder()
        sb.Append("<td colspan='2' style='")
        sb.Append("background-color:" & _
            myControl.BackColor.Name.ToString() & ";' ")

        sb.Append(DesignerRegion.DesignerRegionAttributeName & "='2'>")

        Return sb.ToString()
    End Function

    Public Overrides Function GetEditableDesignerRegionContent( _
            ByVal region As EditableDesignerRegion) As String

        Dim host As IDesignerHost = _
            CType(Component.Site.GetService(GetType(IDesignerHost)), IDesignerHost)

        If (Not host Is Nothing) Then
            Dim template As ITemplate = myControl.View1
            If (region.Name = "Content1") Then
                template = myControl.View2
            End If

            If (Not template Is Nothing) Then
                Return ControlPersister.PersistTemplate(template, host)
            End If

        End If

        Return String.Empty
    End Function

    Public Overrides Sub SetEditableDesignerRegionContent( _
            ByVal region As EditableDesignerRegion, ByVal content As String)

        Dim regionIndex As Int32 = Int32.Parse(region.Name.Substring(7))

        If (content Is Nothing) Then

            If (regionIndex = 0) Then
                myControl.View1 = Nothing
            ElseIf (regionIndex = 1) Then
                myControl.View2 = Nothing
                Return
            End If

            Dim host As IDesignerHost = _
                CType(Component.Site.GetService(GetType(IDesignerHost)),
                    IDesignerHost)

            If (Not host Is Nothing) Then
```

```
                Dim template = ControlParser.ParseTemplate(host, content)

                If (Not template Is Nothing) Then
                    If (regionIndex = 0) Then
                        myControl.View1 = template
                    ElseIf (regionIndex = 1) Then
                        myControl.View2 = template
                    End If
                End If
            End If
        End If
    End Sub
End Class
```

C#

```
public class MultiRegionControlDesigner :
    System.Web.UI.Design.WebControls.CompositeControlDesigner {

    protected int _currentView = 0;
    private MultiRegionControl myControl;
    public override void Initialize(IComponent component)
    {
        base.Initialize(component);
        myControl = (MultiRegionControl)component;
    }

    public override bool AllowResize { get { return true;}}

    protected override void OnClick(DesignerRegionMouseEventArgs e)
    {
        if (e.Region == null)
            return;

        if (e.Region.Name == "Header0" && _currentView != 0) {
            _currentView = 0;
            UpdateDesignTimeHtml();
    }

        if (e.Region.Name == "Header1" && _currentView != 1) {
            _currentView = 1;
            UpdateDesignTimeHtml();
        }
    }

    public override String GetDesignTimeHtml(DesignerRegionCollection regions)
    {
        BuildRegions(regions);
        return BuildDesignTimeHtml();
    }

    protected virtual void BuildRegions(DesignerRegionCollection regions)
    {
        regions.Add(new DesignerRegion(this, "Header0"));
        regions.Add(new DesignerRegion(this, "Header1"));
```

```
            // If the current view is for all, we need another editable region
            EditableDesignerRegion edr0 = new
                EditableDesignerRegion(this, "Content" + _currentView, false);
            edr0.Description = "Add stuff in here if you dare:";
            regions.Add(edr0);

            // Set the highlight, depending upon the selected region
            if (_currentView ==0 || _currentView==1)
                regions[_currentView].Highlight = true;
        }

        protected virtual string BuildDesignTimeHtml()
        {
            StringBuilder sb = new StringBuilder();
            sb.Append(BuildBeginDesignTimeHtml());
            sb.Append(BuildContentDesignTimeHtml());
            sb.Append(BuildEndDesignTimeHtml());

            return sb.ToString();
        }

        protected virtual String BuildBeginDesignTimeHtml()
        {
            // Create the table layout
            StringBuilder sb = new StringBuilder();
            sb.Append("<table ");

            // Styles that we'll use to render for the design-surface
            sb.Append("height='" + myControl.Height.ToString() + "' width='" +
                myControl.Width.ToString() +  "'>");

            // Generate the title or caption bar
            sb.Append("<tr height='25px' align='center' " +
                "style='font-family:tahoma;font-size:10pt;font-weight:bold;'>" +
                "<td style='width:50%' " + DesignerRegion.DesignerRegionAttributeName +
                "='0'>");
            sb.Append("Page-View 1</td>");
            sb.Append("<td style='width:50%' " +
                DesignerRegion.DesignerRegionAttributeName + "='1'>");
            sb.Append("Page-View 2</td></tr>");

            return sb.ToString();
        }

        protected virtual String BuildEndDesignTimeHtml()
        {
            return ("</table>");
        }

        protected virtual String BuildContentDesignTimeHtml()
        {
            StringBuilder sb = new StringBuilder();
            sb.Append("<td colspan='2' style='");
            sb.Append("background-color:" + myControl.BackColor.Name.ToString() +
                ";' ");
            sb.Append(DesignerRegion.DesignerRegionAttributeName + "='2'>");

            return sb.ToString();
        }

        public override string GetEditableDesignerRegionContent
            (EditableDesignerRegion region)
        {
            IDesignerHost host =
                (IDesignerHost)Component.Site.GetService(typeof(IDesignerHost));

            if (host != null) {
                ITemplate template = myControl.View1;

                if (region.Name == "Content1")
                    template = myControl.View2;

                if (template != null)
                    return ControlPersister.PersistTemplate(template, host);
            }

            return String.Empty;
        }

        public override void SetEditableDesignerRegionContent
             (EditableDesignerRegion region, string content)
        {
            int regionIndex = Int32.Parse(region.Name.Substring(7));

            if (content == null)
            {
                if (regionIndex == 0)
                    myControl.View1 = null;
                else if (regionIndex == 1)
                    myControl.View2 = null;
                return;
            }

            IDesignerHost host =
                (IDesignerHost)Component.Site.GetService(typeof(IDesignerHost));

            if (host != null)
            {
                ITemplate template = ControlParser.ParseTemplate(host, content);

                if (template != null)
                {
                    if (regionIndex == 0)
                        myControl.View1 = template;
                    else if (regionIndex == 1)
                        myControl.View2 = template;
                }
            }
        }
    }
```

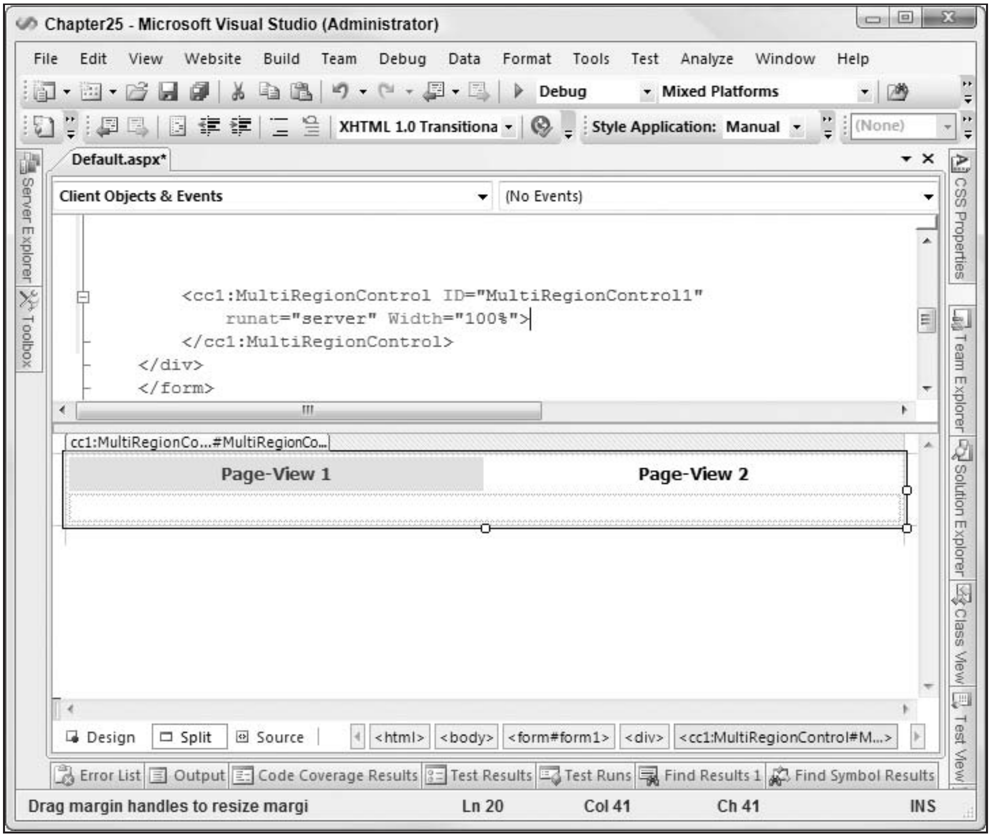
The designer overrides the `GetDesignTimeHtml` method, calling the `BuildRegions` and `BuildDesignTimeHtml` methods to alter the HTML that the control renders to the Visual Studio design surface.

The BuildRegions method creates three design regions in the control, two header regions and an editable content region. The regions are added to the `DesignerRegionCollection`. The `BuildDesignTimeHtml` method calls three methods to generate the actual HTML that will be generated by the control at design time.

The designer class also contains two overridden methods for getting and setting the editable designer region content: `GetEditableDesignerRegionContent` and `SetEditableDesignerRegionContent`. These methods get or set the appropriate content HTML, based on the designer region template that is currently active.

Finally, the class contains an `OnClick` method that it uses to respond to click events fired by the control at design time. This control uses the OnClick event to switch the current region being displayed by the control at design time.

When you add the control to a Web form, you see that you can toggle between the two editable regions, and each region maintains its own content. Figure 26-17 shows what the control looks like on the Visual Studio design surface.


Chapter25 - Microsoft Visual Studio (Administrator)
File
Edit
View
Website
Build
Team
Debug
Data
Format
Tools
Test
Analyze
Window
Help
Debug
Mixed Platforms
XHTML 1.0 Transitiona
Style Application: Manual
(None)
Default.aspx*
Server Explorer
Toolbox
CSS Properties
Team Explorer
Solution Explorer
Class View
Test View
Client Objects & Events
(No Events)
cc1:MultiRegionCo...#MultiRegionCo...
Page-View 1
Page-View 2
Design
Split
Source
<html>
<body>
<form#form1>
<div>
<cc1:MultiRegionControl#M...>
Error List
Output
Code Coverage Results
Test Results
Test Runs
Find Results 1
Find Symbol Results
Drag margin handles to resize margi
Ln 20
Col 41
Ch 41
INS

```
        <cc1:MultiRegionControl ID="MultiRegionControl1"
            runat="server" Width="100%">
        </cc1:MultiRegionControl>
    </div>
    </form>
```

Figure 26-17

As you can see in Figure 26-17, the control contains three separate design regions. When you click design regions 1 or 2, the `OnClick` method in the designer fires and redraws the control on the design surface, changing the template area located in design region 3.

Designer Actions

Another great feature of ASP.NET design-time support is control smart tags. Smart tags give developers using a control quick access to common control properties. Smart tags are actually a new and improved implementation of the Designer Verbs functionality that was available in ASP.NET 1.0.

To add menu items to a server control's smart tag, you create a new class that inherits from the `DesignerActionList` class. The `DesignerActionList` contains the list of designer action items that are displayed by a server control. Classes that derive from the `DesignerActionList` class can override the `GetSortedActionItems` method, creating their own `DesignerActionItemsCollection` object to which designer action items can be added.

You can add several different types of `DesignerActionItems` types to the collection:

- `DesignerActionTextItem`
- `DesignerActionHeaderItem`
- `DesignerActionMethodItem`
- `DesignerActionPropertyItem`

Listing 26-44 shows a control designer class that contains a private class deriving from `DesignerActionList`.

Listing 26-44: Adding designer actions to a control designer

VB

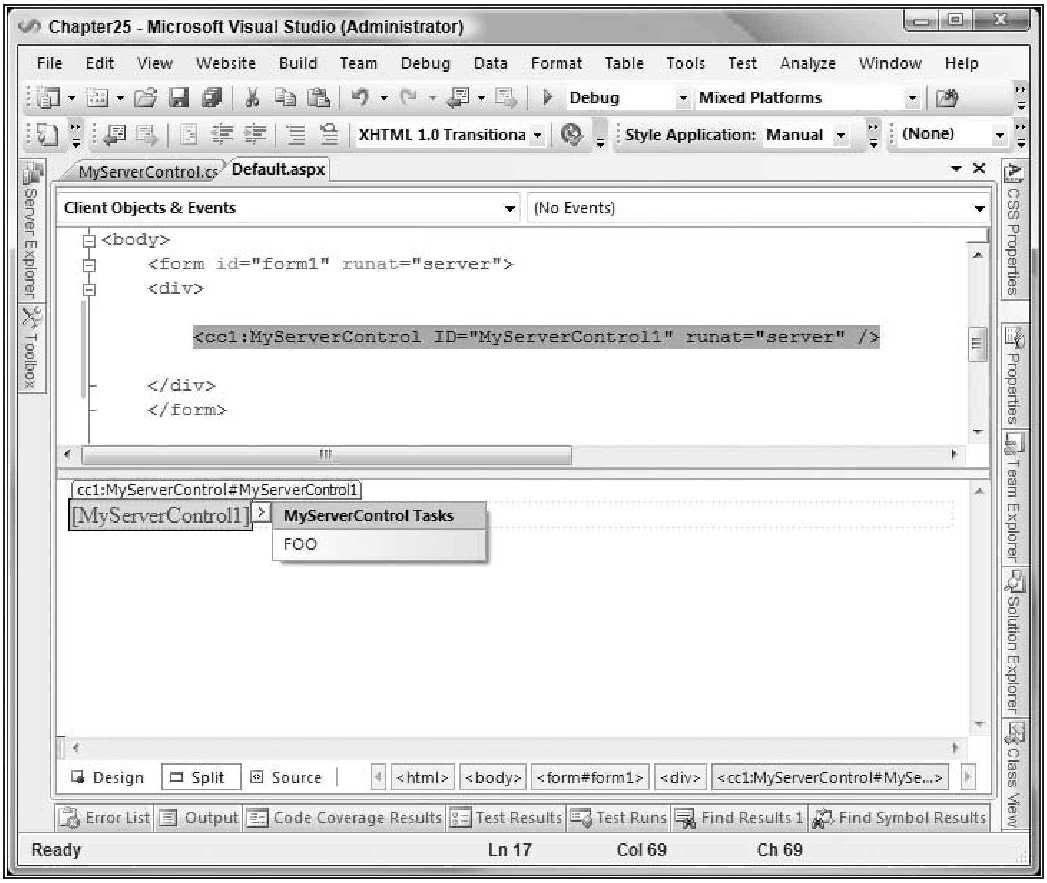
```
Imports System.Web.UI.Design
Imports System.ComponentModel.Design

Public Class TestControlDesigner
    Inherits ControlDesigner

    Public Overrides ReadOnly Property ActionLists() _
            As DesignerActionListCollection
        Get
            Dim lists As New DesignerActionListCollection()
            lists.AddRange(MyBase.ActionLists)
            lists.Add(New TestControlList(Me))
            Return lists
        End Get
    End Property

    Private NotInheritable Class TestControlList
        Inherits DesignerActionList

        Public Sub New(ByVal c As TestControlDesigner)
            MyBase.New(c.Component)
        End Sub
        Public Overrides Function GetSortedActionItems() _
                As DesignerActionItemCollection

            Dim c As New DesignerActionItemCollection()
            c.Add(New DesignerActionTextItem("FOO", "FOO"))
            Return c
        End Function
    End Class

End Class
```

C#

```
using Syste;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.WebControls;
using System.Web.UI.Design;

public class TestControlDesigner : ControlDesigner
{
    public override DesignerActionListCollection ActionLists
    {
        get
        {
            DesignerActionListCollection actionLists =
                new DesignerActionListCollection();
            actionLists.AddRange(base.ActionLists);
            actionLists.Add(new TestControlList(this));
            return actionLists;
        }
    }

    private sealed class TestControlList : DesignerActionList
    {
        public TestControlList(TestControlDesigner c) : base(c.Component)
        {
        }

        public override DesignerActionItemCollection GetSortedActionItems()
        {
            DesignerActionItemCollection c = new DesignerActionItemCollection();
            c.Add(new DesignerActionTextItem("FOO", "FOO"));
            return c;
        }
    }
}
```

The control designer class overrides the `ActionsLists` property. The property creates an instance of the `TextControlList` class, which derives from `DesignerActionList` and overrides the `GetSortedActionItems`method. The method creates a new `DesignerActionListCollection`, and a

`DesignerActionTextItem` is added to the collection (see Figure 26-18). The `DesignerActionTextItem` class allows you to add text menu items to the smart tag.


Chapter25 - Microsoft Visual Studio (Administrator)
File Edit View Website Build Team Debug Data Format Table Tools Test Analyze Window Help
Debug
Mixed Platforms
XHTML 1.0 Transitiona
Style Application: Manual
(None)
MyServerControl.cs
Default.aspx
Client Objects & Events
(No Events)
Server Explorer
Toolbox
CSS Properties
Properties
Team Explorer
Solution Explorer
Class View
cc1:MyServerControl#MyServerControl1
[MyServerControl1]
MyServerControl Tasks
FOO
Design
Split
Source
<html> <body> <form#form1> <div> <cc1:MyServerControl#MySe...>
Error List
Output
Code Coverage Results
Test Results
Test Runs
Find Results 1
Find Symbol Results
Ready
Ln 17
Col 69
Ch 69

```
<body>
    <form id="form1" runat="server">
    <div>

        <cc1:MyServerControl ID="MyServerControl1" runat="server" />

    </div>
    </form>
```

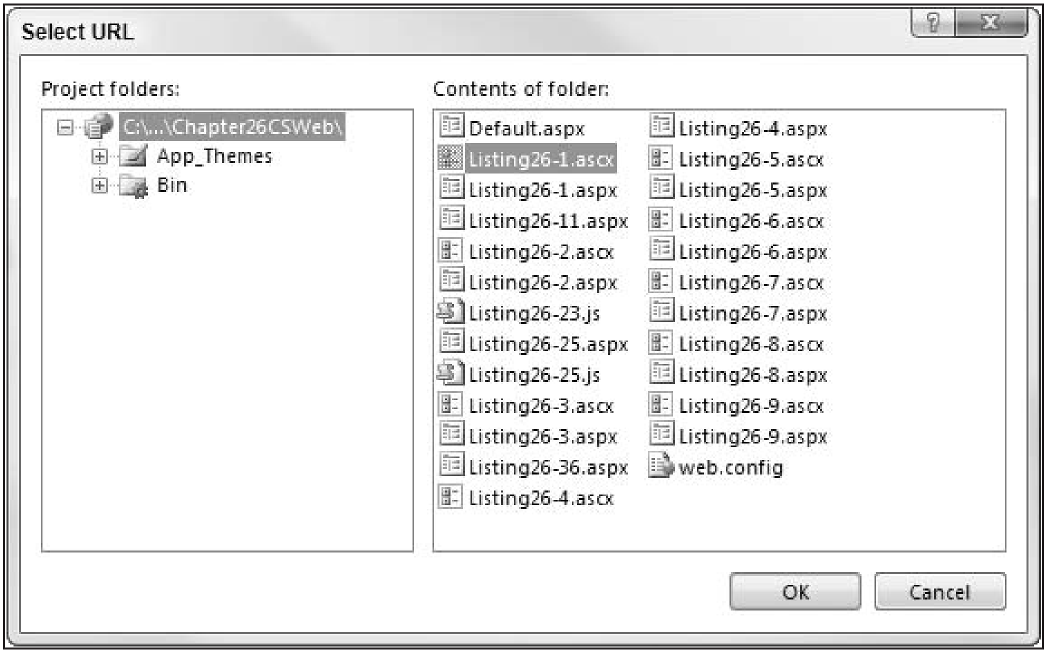
Figure 26-18

As shown in Figure 26-18, when you add the control to a Web page, the control now has a smart tag with the `DesignerActionTextItem` class as content.

UI Type Editors

A UI type editor is a way to provide users of your controls with a custom interface for editing properties directly from the Property Browser. One type of UI type editor you might already be familiar with is the Color Picker you see when you want to change the `ForeColor` attribute that exists on most ASP.NET controls. ASP.NET provides a wide variety of in-box UI type editors that make it easy to edit more complex property types. The easiest way to find what UI type editors are available in the .NET Framework is to search for types derived from the `UITypeEditor` class in the MSDN Library help.

After you find the type editor you want to use on your control property, you simply apply the UI Type Editor to the property using the `Editor` attribute. Listing 26-45 shows how to do this.



C:\...\Chapter26CSWeb\
App_Themes
Bin

Default.aspx
Listing26-1.ascx
Listing26-1.aspx
Listing26-11.aspx
Listing26-2.ascx
Listing26-2.aspx
Listing26-23.js
Listing26-25.aspx
Listing26-25.js
Listing26-3.ascx
Listing26-3.aspx
Listing26-36.aspx
Listing26-4.ascx
Listing26-4.aspx
Listing26-5.ascx
Listing26-5.aspx
Listing26-6.ascx
Listing26-6.aspx
Listing26-7.ascx
Listing26-7.aspx
Listing26-8.ascx
Listing26-8.aspx
Listing26-9.ascx
Listing26-9.aspx
web.config

Figure 26-19

Listing 26-45: Adding a UI type editor to a control property

VB

```
<Bindable(True), Category("Appearance"), DefaultValue(""), _
Editor( _
        GetType(System.Web.UI.Design.UrlEditor), _
        GetType(System.Drawing.Design.UITypeEditor))> _
Public Property Url() As String
    Get
        Return _url
    End Get
    Set(ByVal value As String)
        _url = value
    End Set
End Property
```

C#

```
[Bindable(true)]
[Category("Appearance")]
[DefaultValue("")]
[Editor(typeof(System.Web.UI.Design.UrlEditor),
    typeof(System.Drawing.Design.UITypeEditor))]
public string Url
{
    get
    {
        return url;
    }
```

```
        set
        {
            url = value;
        }
    }
```

In this sample, you have created a `Url` property for a control. Because you know this property will be a URL, you want to give the control user a positive design-time experience. You can use the UrlEditor type editor to make it easier for users to select a URL. Figure 26-19 shows the Url Editor that appears when the user edits the control property.

Summary

In this chapter, you learned a number of ways you can create reusable, encapsulated chunks of code. You first looked at user controls, the simplest form of control creation. You learned how to create user controls and how you can make them interact with their host Web pages. Creating user controls is quite easy, but they lack the portability of other control-creation options.

Then, you saw how you can create your own custom server controls. You looked at many of the tools you can create by writing custom server controls. These range from tools for emitting HTML and creating CSS styles and JavaScript to those applying themes. The chapter also discussed the type of server controls you can create, ranging from server controls that simply inherit from the `WebControl` class to templated controls that give users of the control the power to define the display of the server control.

Finally, you looked at ways you can give the users of your server control a great design-time experience by providing them with TypeConvertors, design surface interactions, and custom property editors in your server control.

27

Modules and Handlers

Sometimes, just creating dynamic Web pages with the latest languages and databases does not give you, the developer, enough control over an application. At times, you need to be able to dig deeper and create applications that can interact with the Web server itself. You want to be able to interact with the low-level processes, such as how the Web server processes incoming and outgoing HTTP requests.

Before ASP.NET, to get this level of control using IIS, you were forced to create ISAPI extensions or filters. This proved to be quite a daunting and painful task for many developers because creating ISAPI extensions and filters required knowledge of C/C++ and knowledge of how to create native Win32 DLLs. Thankfully, in the .NET world, creating these types of low-level applications is really no more difficult than most other tasks you would normally perform. This chapter looks at two methods of manipulating how ASP.NET processes HTTP requests, the HttpModule, and the HttpHandler. Each method provides a unique level of access to the underlying processing of ASP.NET and can be a powerful tool for creating Web applications.

Processing HTTP Requests

Before starting to write Handlers or Modules, it's helpful to know how IIS and ASP.NET normally process incoming HTTP requests and what options you have for plugging custom logic into those requests. IIS is the basic endpoint for incoming HTTP requests. At a very high level, its job is to listen for and validate incoming HTTP requests. Then it routes them to the appropriate module for processing and returns any results to the original requestor. ASP.NET is one of the modules that IIS may pass requests to for processing. However, exactly how that processing happens and how you can integrate your own logic into the pipeline differs based on the version of IIS you are using.

IIS 5/6 and ASP.NET

If you are using IIS 5 or IIS 6, the HTTP request processing pipeline is fairly black box to a managed code developer. IIS basically treats ASP.NET as one of the modules that it can pass requests to for processing rather than as an integrated part of the IIS request processing pipeline. Figure 27-1 shows the basic request processing pipeline of IIS 5/6 and ASP.NET.

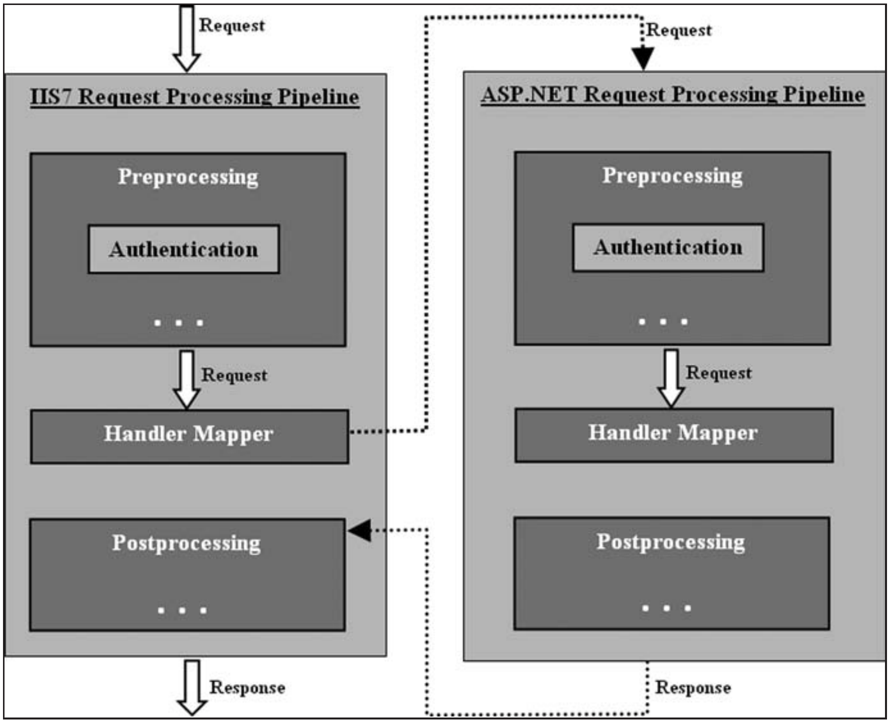

Request
IIS7 Request Processing Pipeline
Preprocessing
Authentication
. . .
Request
Handler Mapper
Postprocessing
. . .
Response
Request
ASP.NET Request Processing Pipeline
Preprocessing
Authentication
. . .
Request
Handler Mapper
Postprocessing
. . .
Response

Figure 27-1

As you can see, the IIS and ASP.NET request pipelines are very similar and several tasks, such as authentication, are even duplicated between the two pipelines. Furthermore, while you can write Handlers and Modules using managed code, they are still processed in the isolated context of the ASP.NET process. If you wanted to integrate deeper into the IIS pipeline you are forced to create modules using native code.

IIS 7 and ASP.NET

Starting with IIS 7, the request processing pipeline in IIS has been completely re-architected using an open and highly extensible module based system. Now, instead of IIS seeing ASP.NET as a separate entity, ASP.NET has been deeply integrated into the IIS request processing pipeline. As shown in Figure 27-2, the request processing pipeline has been streamlined to eliminate duplicate processes and to allow you to integrate managed modules in the pipeline.

Because ASP.NET modules are first-class citizens, you can place them at any point in the pipeline, or even completely replace existing modules with your own custom functionality. Features that previously required you to write custom ISAPI modules in unmanaged code can now simply be replaced by managed code modules containing your logic. If you are interested in learning more about the integration of ASP.NET and IIS 7, check out the Wrox title *Professional IIS 7 and ASP.NET Integrated Programming* by Shahram Khosravi (Wiley Publishing, Inc., 2007).

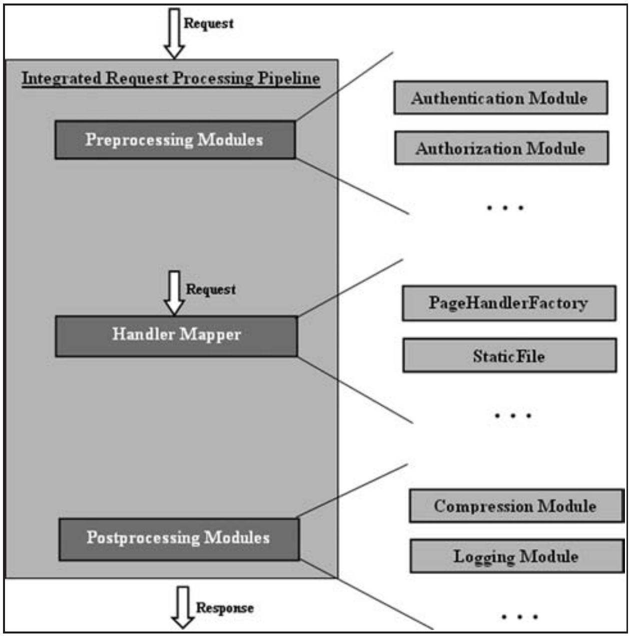

Request
Integrated Request Processing Pipeline
Preprocessing Modules
Authentication Module
Authorization Module
. . .
Request
Handler Mapper
PageHandlerFactory
StaticFile
. . .
Compression Module
Postprocessing Modules
Logging Module
Response
. . .

Figure 27-2

ASP.NET Request Processing

Regardless of the IIS version, the basic HTTP request pipeline model has two core mechanisms for handling requests: HttpModules and HttpHandlers. ASP.NET uses those two mechanisms to process incoming ASP.NET requests, generate a response, and return that response to the client. In fact, you are probably already familiar with HttpModules and HttpHandlers — although you might not know it. If you have ever used the Inbox caching or the authentication features of ASP.NET, you have used several different HttpModules. Additionally, if you have ever served up an ASP.NET application, even something as simple as a *Hello World* Web page and viewed it in a browser, you have used an HttpHandler. ASP.NET uses handlers to process and render ASPX pages and other file extensions. Modules and handlers allow you to plug into the request-processing pipeline at different points and interact with the actual requests being processed by IIS.

As you can see in both Figures 27-1 and 27-2, ASP.NET passes each incoming request through a layer of preprocessing HttpModules in the pipeline. ASP.NET allows multiple modules to exist in the pipeline for each request. After the incoming request has passed through each module, it is passed to the HttpHandler, which serves the request. Notice that although a single request may pass through many different modules, it can be processed by one handler only. The handler is generally responsible for creating a response to the incoming HTTP request. After the handler has completed execution and generated a response, the response is passed back through a series of post-processing modules, before it is returned to the client.

You should now have a basic understanding of the IIS and ASP.NET request pipeline — and how you can use HttpModules and HttpHandlers to interact with the pipeline. The following sections take an in-depth look at each of these.

HttpModules

HttpModules are simple classes that can plug themselves into the request-processing pipeline. They do this by hooking into a handful of events thrown by the application as it processes the HTTP request. To create an HttpModule, you simply create a class that derives from the `System.Web.IHttpModule` interface. This interface requires you to implement two methods: `Init` and `Dispose`. Listing 27-1 shows the class stub created after you implement the `IHttpModule` interface.

Listing 27-1: Implementing the IHttpModule Interface

VB

```
Imports Microsoft.VisualBasic
Imports System.Web

Namespace Demo

    Public Class SimpleModule
        Implements IHttpModule

        Public Overridable Sub Init(ByVal context As HttpApplication) _
                Implements IHttpModule.Init

        End Sub

        Public Overridable Sub Dispose() Implements IHttpModule.Dispose

        End Sub
    End Class

End Namespace
```

C#

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web;

namespace Demo
{
    class SimpleModule : IHttpModule
    {
        #region IHttpModule Members

        public void Dispose()
        {
            throw new Exception("The method or operation is not implemented.");
        }
```

```
            public void Init(HttpApplication context)
            {
                throw new Exception("The method or operation is not implemented.");
            }

            #endregion
        }
    }
```

The `Init` method is the primary method you use to implement functionality. Notice that it has a single method parameter, an `HttpApplication` object named `context`. This parameter gives you access to the current `HttpApplication` context, and it is what you use to wire up the different events that fire during the request processing. The following table shows the events that you can register in the `Init` method.

Event Name	**Description**
`AcquireRequestState`	Raised when ASP.NET runtime is ready to acquire the Session State of the current HTTP request.
`AuthenticateRequest`	Raised when ASP.NET runtime is ready to authenticate the identity of the user.
`AuthorizeRequest`	Raised when ASP.NET runtime is ready to authorize the user for the resources user is trying to access.
`BeginRequest`	Raised when ASP.NET runtime receives a new HTTP request.
`Disposed`	Raised when ASP.NET completes the processing of HTTP request.
`EndRequest`	Raised just before sending the response content to the client.
`Error`	Raised when an unhandled exception occurs during the processing of HTTP request.
`PostRequestHandlerExecute`	Raised just after HTTP handler finishes execution.
`PreRequestHandlerExecute`	Raised just before ASP.NET begins executing a handler for the HTTP request. After this event, ASP.NET forwards the request to the appropriate HTTP handler.
`PreSendRequestContent`	Raised just before ASP.NET sends the response contents to the client. This event allows you to change the contents before it gets delivered to the client. You can use this event to add the contents, which are common in all pages, to the page output. For example, a common menu, header, or footer.
`PreSendRequestHeaders`	Raised just before ASP.NET sends the HTTP response headers to the client. This event allows you to change the headers before they get delivered to the client. You can use this event to add cookies and custom data into headers.

Modifying HTTP Output

Take a look at some examples of using HttpModules. The first example shows a useful way of modifying the HTTP output stream before it is sent to the client. This can be a simple and useful tool if you want to add text to each page served from your Web site, but you do not want to modify each page. For the first example, create a Web project in Visual Studio and add a class to the App_Code directory. The code for this first module is shown in Listing 27-2.

Listing 27-2: Altering the output of an ASP.NET Web page

VB

```
Imports Microsoft.VisualBasic
Imports System.Web

Namespace Demo

    Public Class AppendMessage
        Implements IHttpModule

        Dim WithEvents _application As HttpApplication = Nothing

        Public Overridable Sub Init(ByVal context As HttpApplication) _
                Implements IHttpModule.Init
            _application = context
        End Sub

        Public Overridable Sub Dispose() Implements IHttpModule.Dispose

        End Sub

        Public Sub context_PreSendRequestContent(ByVal sender As Object, _
                ByVal e As EventArgs) Handles _application.PreSendRequestContent

            'alter the outgoing content by adding a HTML comment.
            Dim message As String = "<!-- This page has been post processed at " & _
                                    System.DateTime.Now.ToString() & _
                                    " by a custom HttpModule.-->"

            _application.Context.Response.Output.Write(message)

        End Sub
    End Class

End Namespace
```

C#

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web;

namespace Demo
{
    public class AppendMessage : IHttpModule
    {
        private HttpContext _current = null;

        #region IHttpModule Members

        public void Dispose()
        {
            throw new Exception("The method or operation is not implemented.");
        }

        public void Init(System.Web.HttpApplication context)
        {
            _current = context.Context;

            context.PreSendRequestContent +=
                new EventHandler(context_PreSendRequestContent);
        }

        void context_PreSendRequestContent(object sender, EventArgs e)
        {
            //alter the outgoing content by adding a HTML comment.
            string message = "<!-- This page has been post processed at " +
                             System.DateTime.Now.ToString() +
                             " by a custom HttpModule.-->";

            _current.Response.Output.Write(message);
        }

        #endregion
    }
}
```

You can see that the class stub from Listing 27-2 is expanded here. In the `Init` method, you register the `PreSendRequestContent` event. This event fires right before the content is sent to the client, and you have one last opportunity to modify it.

In the `PreSendRequestContent` handler method, you simply create a string containing an HTML comment that contains the current time. You take this string and write it to the current HTTP requests output stream. The HTTP request is then sent back to the client.

In order to use this module, you must let ASP.NET know that you want to include the module in the request-processing pipeline. You do this is by modifying the `web.config` to contain a reference to the module. Listing 27-3 shows how you can add an httpModules section to your `web.config`.

Listing 27-3: Adding the httpModule configuration to web.config

```
<configuration>
    <system.web>
      <httpModules>
        <add name="AppendMessage" type="Demo.AppendMessage, App_code" />
      </httpModules>
    </system.web>
</configuration>
```

The generic format of the httpModules section is

```
<httpModules>
  <add name="modulename" type="namespace.classname, assemblyname" />
</httpModules>
```

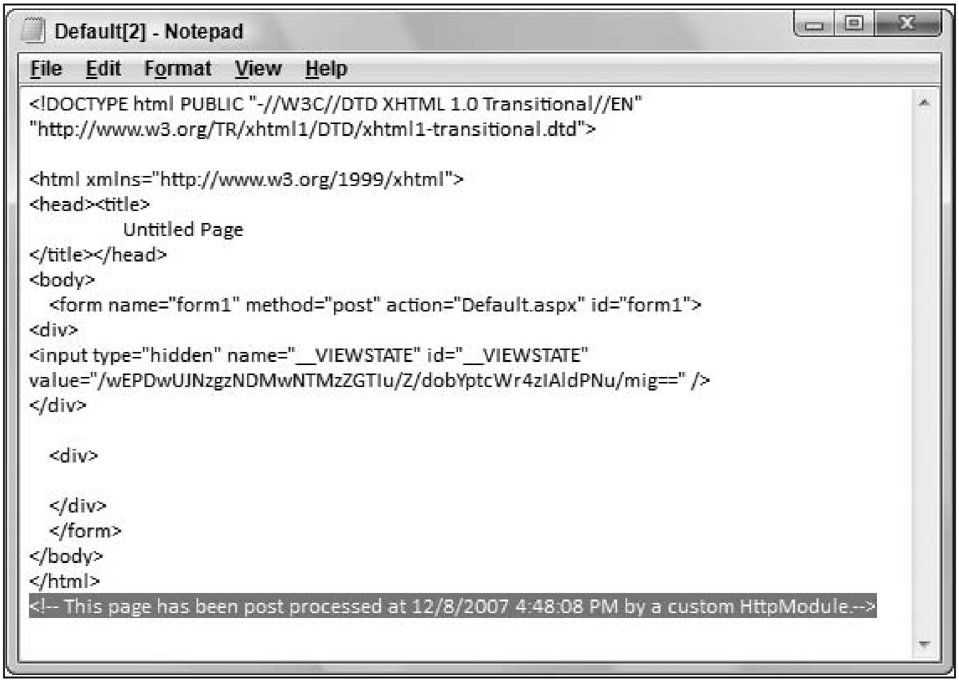
If you are deploying your application to an IIS 7 server, you must also add the module configuration to the <system.webServer> configuration section.

```
<modules>
    <add name="AppendMessage" type="Demo.AppendMessage, App_code"/>
</modules>
```

If you have created your HttpModule in the `App_Code` directory of an ASP.NET web site, you might wonder how you know what the `assemblyname` value should be, considering ASP.NET now dynamically compiles this code at runtime. The solution is to use the text `App_Code` as the assembly name. This tells ASP.NET that your module is located in the dynamically created assembly.

You can also create HttpModules as a separate class library in which case you simply use the assembly name of the library.

After you have added this section to your `web.config` file, simply view one of the Web pages from your project in the browser. When you view the page in the browser, you should not notice any difference. But if you view the source of the page, notice the comment you added at the bottom of the HTML. Figure 27-3 shows what you should see when you view the page source.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
        Untitled Page
</title></head>
<body>
  <form name="form1" method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUJNzgzNDMwNTMzZGTIu/Z/dobYptcWr4zIAldPNu/mig==" />
</div>

  <div>

  </div>
  </form>
</body>
</html>
<!-- This page has been post processed at 12/8/2007 4:48:08 PM by a custom HttpModule.-->
```

Figure 27-3

URL Rewriting

Another interesting use of an HttpModule is to perform URL rewriting. *URL rewriting* is a technique that allows you to intercept the HTTP request and change the path that was requested to an alternative one. This can be very useful for creating pseudo Web addresses that simplify a URL for the user. For example, the MSDN Library is well known for its extremely long and cryptic URL paths, such as

```
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/
frlrfSystemWebIHttpModuleClassTopic.asp
```

The problem with this URL is that it is not easy to remember; and even if you do somehow remember it, it is very difficult to type into the browser's Address field. URL rewriting allows you to create friendly URLs that you can parse and redirect to the appropriate resource. The MSDN Library now uses URL rewriting to create friendly URLs. Instead of the cryptic URL you saw previously, you can now use the following URL to access the same resource:

```
http://msdn2.microsoft.com/library/system.web.ihttpmodule.aspx
```

This URL is much shorter, easier to remember, and easier to type into a browser's Address field. You can create your own URL rewriter module to learn how this is done.

To demonstrate this, you create three new Web pages in your project. The first Web page is used to construct a URL using two text boxes. The second serves as the page that accepts the unfriendly querystring, like the MSDN URL shown previously. The third page is used to trick IIS into helping you serve the request. Shortly, we talk about this trick and how to get around it.

Listing 27-4 shows the first Web page you add to the project; it's called `friendlylink.aspx`.

Listing 27-4: The friendlylink.aspx Web page

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <a href="John/Smith/trickiis.aspx">Click this friendly link</a>
    </div>
    </form>
</body>
</html>
```

As you can see, you simply created a hyperlink that links to a friendly, easily remembered URL.

Now, create the second page called `unfriendly.aspx`. This is the page that the handle actually executes when a user clicks the hyperlink in the `friendlylink.aspx` page. Listing 27-5 shows how to create `unfriendly.aspx`.

Listing 27-5: The unfriendly.aspx Web page

VB

```
<%@ Page Language="VB" %>
```

Continued

```
<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, _
            ByVal e As System.EventArgs) Handles Me.Load
        Label1.Text = Request("firstname").ToString() & _
            " " & Request("lastname").ToString()
    End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Unfriendly Web Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
        Welcome to the unfriendly URL page <asp:Label ID="Label1"
             runat="server" Text="Label"></asp:Label>
        </div>
    </form>
</body>
</html>
```

C#

```
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Label1.Text = Request["firstname"].ToString() +
            " " + Request["lastname"].ToString();
    }
</script>
```

Next, you create the directory and file that the hyperlink in `friendlyurl.aspx` points to. The `trickiis.aspx` page can simply be an empty Web page because you are not really going to execute it.

Finally, you create a new module that parses the request path and rewrites the URL to the page you want to execute. To do this, create another class in the `App_Code` directory called `SimpleRewriter`. Listing 27-6 shows the code for this.

Listing 27-6: A sample URL rewriting HttpModule

VB

```
Imports Microsoft.VisualBasic
Imports System.Web

Namespace Demo
    Public Class SimpleRewriter
        Implements System.Web.IHttpModule

        Dim WithEvents _application As HttpApplication = Nothing

        Public Overridable Sub Init(ByVal context As HttpApplication) _
                Implements IHttpModule.Init
            _application = context
        End Sub
        Public Overridable Sub Dispose() Implements IHttpModule.Dispose

        End Sub

        Public Sub context_BeginRequest(ByVal sender As Object, _
            ByVal e As EventArgs) Handles _application.BeginRequest

            Dim requesturl As String = _
                _application.Context.Request.Path.Substring(0, _
                _application.Context.Request.Path.LastIndexOf("/"c))

            'Here is where we parse the original request url to determine
            ' the querystring parameters for the unfriendly url
            Dim parameters() As String = _
                requesturl.Split(New [Char]() {"/"c}, _
                    StringSplitOptions.RemoveEmptyEntries)

            If (parameters.Length > 1) Then
                Dim firstname As String = parameters(1)
                Dim lastname As String = parameters(2)

                'Rewrite the request path
                _application.Context.RewritePath("~/unfriendly.aspx?firstname=" & _
                    firstname & "&lastname=" & lastname)
            End If
        End Sub
    End Class

End Namespace
```

C#

```
using System.Web;

namespace Demo
{
    public class SimpleRewriter: System.Web.IHttpModule
    {

        HttpApplication _application = null;

        public void Init(HttpApplication context)
        {
            context.BeginRequest+=new System.EventHandler(context_BeginRequest);
            _application = context;
        }

        public void Dispose()
        {
        }

        private void context_BeginRequest(object sender, System.EventArgs e)
        {
            string requesturl =
                _application.Context.Request.Path.Substring(0,
```

Continued

```
                    _application.Context.Request.Path.LastIndexOf("//")
                );

            //Here is where we parse the original request url to determine
            //the querystring parameters for the unfriendly url
            string[] parameters = requesturl.Split(new char[] {'/'});

            if (parameters.Length > 1)
            {
                string firstname = parameters[1];
                string lastname = parameters[2];

                //Rewrite the request path
                _application.Context.RewritePath("~/unfriendly.aspx?firstname=" +
                    firstname + "&lastname=" + lastname);
            }
        }
    }
}
```

As you can see from the listing, in this sample you use the BeginRequest event in the HttpModule to parse the incoming HTTP request path and create a new URL that you execute. Normally, when you click the hyperlink on `friendlyurl.aspx`, an HTTP request is sent to the server for execution and then IIS returns the page asked for in the hyperlink. In this case, you make a request for this page:

```
http://localhost:1234/WebProject1/John/Smith/trickiis.aspx
```

But, because you put the HttpModule in the request-processing pipeline, you can modify the HTTP request and change its behavior. The code in the `BeginRequest` method of the module parses the request path to create a querystring that the `unfriendly.aspx` page can understand and execute. So when you execute the code in the listing, you convert the original path into the following:

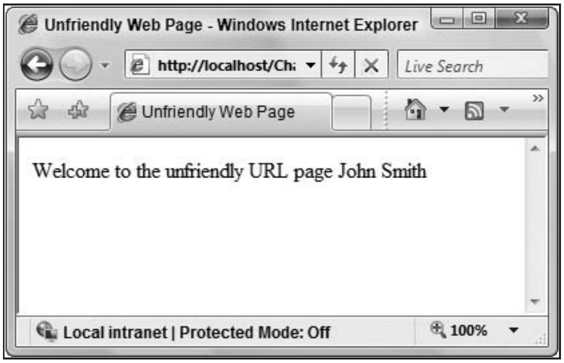
```
http://localhost:1234/WebProject1/unfriendly.aspx?var1=John&var2=Smith
```

This URL is, as the page name states, not very friendly; and the user is less likely to remember and be able to type this URL. Finally, the module uses the `RewritePath` method to tell ASP.NET that you want to rewrite the path to execute for this request.

After you have completed creating the code for this sample, try loading `friendlylink.aspx` into a browser. When you click the hyperlink on the page, you should notice two things. First, notice that the URL in the browser's address bar shows that you have been served the page you requested, `trickiis.aspx`, but the contents of the page show that you are actually served `unfriendly.aspx`. Figure 27-4 shows what the browser looks like.

IIS WildCards

There is, however, a drawback to this type of URL rewriting. When IIS receives a request to serve a resource, it first checks to see if the resource exists. If the resource does exist, the request is passed to the appropriate handler; in this case, the handler is ASP.NET, which processes the request. IIS then returns the results to the client. However, if the requested resource does not exist, IIS returns a `404 File Not Found` error to the client. It never hands the request to ASP.NET. In order for a module to execute, you must create an endpoint that actually exists on the Web server.


Unfriendly Web Page - Windows Internet Explorer
http://localhost/Ch
Live Search
Unfriendly Web Page
Local intranet | Protected Mode: Off
100%

Welcome to the unfriendly URL page John Smith

Figure 27-4

In the case of the previous example, you had to actually create the `/WebProject1/John/Smith/trickiis.aspx` path so that IIS would know which handler to give the request to. Had you not created the path, IIS would have simply returned a 404 error. Unfortunately, having to create the physical paths can be a problem if you find you are creating a large number of directories or resources just to enable URL rewriting. Thankfully, however, a solution to this problem exists in IIS: wildcards. *Wildcards* allow you to tell IIS that it should use a particular handler to process all incoming requests, regardless of the requested paths or file extensions. In the previous example, adding a wildcard would keep you from having to actually create the dummy end point for the module.

Adding Wildcards in IIS 5

To add a wildcard mapping in IIS 5, start by opening the IIS Management Console. After the console is open, right-click on the Web site or virtual directory you want to modify and select the Properties option from the context menu. When the Properties dialog box opens, select the Configuration button from the Home Directory Tab. The Configuration dialog is where you can create new or modify existing application extension mappings. If you take a moment to look at the existing mappings, you see that most of the familiar file extensions (such as `.aspx`, `.asp`, and `.html`) are configured.

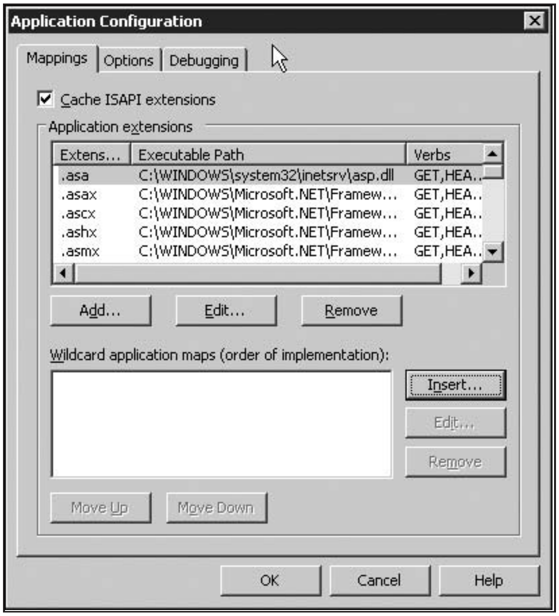
To add the wildcard that you need to create a new mapping, click the Add button. The Add Application Extension Mapping dialog box is shown in Figure 27-5. You create a mapping that directs IIS to use the ASP.NET ISAPI DLL to process every incoming request. To do this, simply put the full path to the ISAPI DLL (usually something such as `C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322\aspnet_isapi.dll`) in the Executable field. For the extension, simply use .*, which indicates any file extension.

Additionally, you uncheck the Check That File Exists check box to tell IIS not to check whether the requested file exists before processing (because you know that it doesn't).

Now you don't have to add the stub file to your Web site. IIS will pass any request that it receives to ASP.NET for processing, regardless of whether the file exists.

Adding Wildcards in IIS 6

Adding Wildcards in IIS 6 is similar to adding wildcards in IIS 5. Open the IIS Management Console, and then open the Properties dialog box for the Web site or virtual directory you want to modify. Next, click the Configuration button on the Home Directory tab.


Application Configuration
Mappings
Options
Debugging
Application extensions
Add...
Edit...
Remove
Wildcard application maps (order of implementation):
Insert...
Edit...
Remove
Move Up
Move Down
OK
Cancel
Help

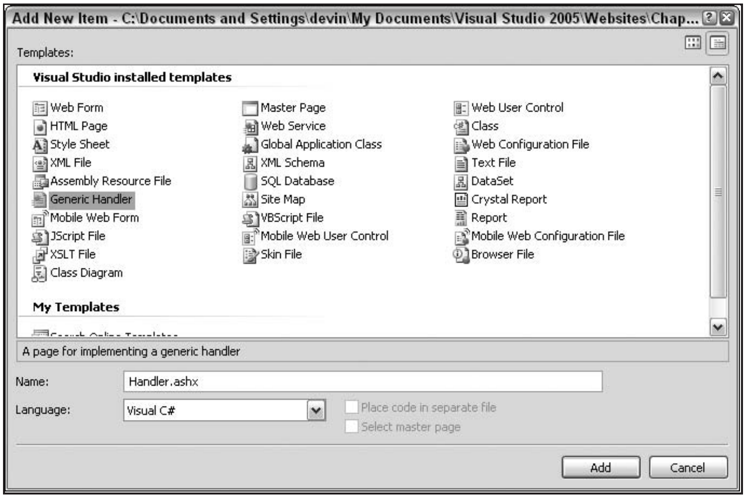
☑ Cache ISAPI extensions

Extens...	Executable Path	Verbs
.asa	C:\WINDOWS\system32\inetsrv\asp.dll	GET,HEA...
.asax	C:\WINDOWS\Microsoft.NET\Framew...	GET,HEA...
.ascx	C:\WINDOWS\Microsoft.NET\Framew...	GET,HEA...
.ashx	C:\WINDOWS\Microsoft.NET\Framew...	GET,HEA...
.asmx	C:\WINDOWS\Microsoft.NET\Framew...	GET,HEA...

Figure 27-5

The Application Extension Configuration dialog in IIS is slightly different. Wildcard application maps now have their own separate listing, as shown in Figure 27-6.

To add a wildcard mapping, click the Insert button, add the path to the ASP.NET ISAPI DLL, and make sure the Verify That File Exists check box is unchecked.


Add New Item - C:\Documents and Settings\devin\My Documents\Visual Studio 2005\Websites\Chap...
Templates:
Visual Studio installed templates
Web Form
HTML Page
Style Sheet
XML File
Assembly Resource File
Generic Handler
Mobile Web Form
JScript File
XSLT File
Class Diagram
Master Page
Web Service
Global Application Class
XML Schema
SQL Database
Site Map
VBScript File
Mobile Web User Control
Skin File
Web User Control
Class
Web Configuration File
Text File
DataSet
Crystal Report
Report
Mobile Web Configuration File
Browser File
My Templates
Add
Cancel

A page for implementing a generic handler

Name: Handler.ashx

Language: Visual C#

☐ Place code in separate file

☐ Select master page

Figure 27-6

HttpHandlers

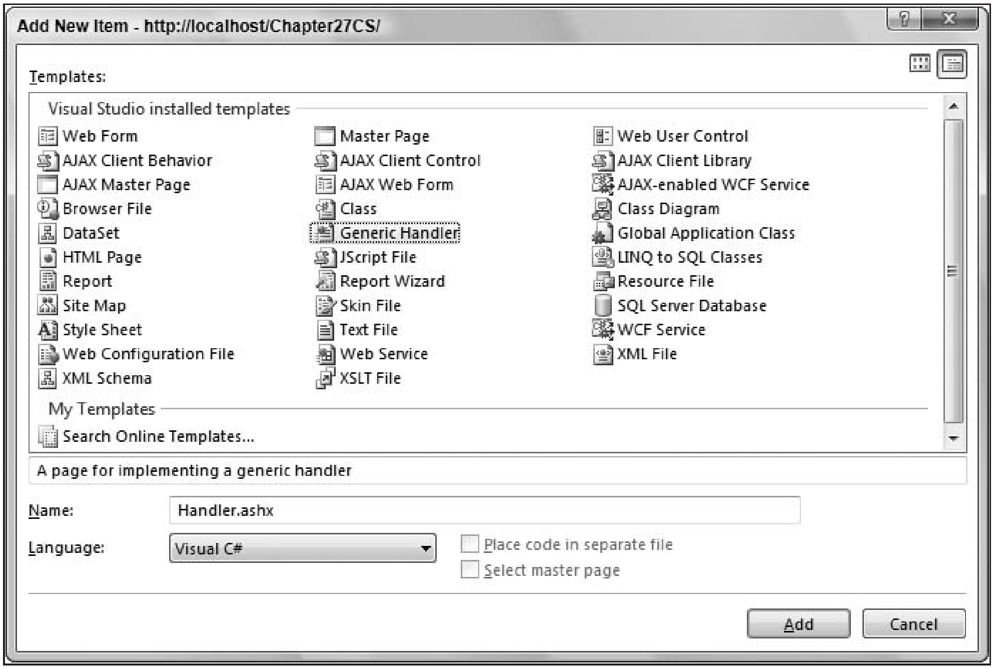
HttpHandlers differ from HttpModules, not only because of their positions in the request-processing pipeline (refer to Figure 27-3), but also because they must be mapped to a specific file extension. Handlers are the last stop for incoming HTTP requests and are ultimately the point in the request-processing pipeline that is responsible for serving up the requested content, be it an ASPX page, HTML, plain text, or an image. Additionally, HttpHandlers can offer significant performance gains.

In this section, we demonstrate two different ways to create a simple HttpHandler that you can use to serve up dynamic images. First, you look at creating an HttpHandler using an ASHX file extension. Then you learn how you get even more control by mapping your HttpHandler to a custom file extension using IIS.

Generic Handlers

In early versions of Visual Studio, HttpHandlers were somewhat hard to understand and create because little documentation was included to help developers understand handlers. In addition Visual Studio did not provide any friendly methods for creating them.

Since Visual Studio 2005 however this has changed with Visual Studio now providing a standard template for HttpHandlers to help you get started. To add an HttpHandler to your project, you simply select the Generic Handler file type from the Add New Item dialog. Figure 27-7 shows this dialog box with the file type selected.


Add New Item - http://localhost/Chapter27CS/
Templates:
Visual Studio installed templates
Web Form
AJAX Client Behavior
AJAX Master Page
Browser File
DataSet
HTML Page
Report
Site Map
Style Sheet
Web Configuration File
XML Schema
Master Page
AJAX Client Control
AJAX Web Form
Class
Generic Handler
JScript File
Report Wizard
Skin File
Text File
Web Service
XSLT File
Web User Control
AJAX Client Library
AJAX-enabled WCF Service
Class Diagram
Global Application Class
LINQ to SQL Classes
Resource File
SQL Server Database
WCF Service
XML File
My Templates
Search Online Templates...
Add
Cancel

A page for implementing a generic handler

Name: Handler.ashx

Language: Visual C#

☐ Place code in separate file

☐ Select master page

Figure 27-7

You can see that when you add the Generic Handler file to your project, it adds a file with an `.ashx` extension. The `.ashx` file extension is the default HttpHandler file extension set up by ASP.NET. Remember that HttpHandlers must be mapped to a unique file extension, so by default ASP.NET uses the `.ashx` extension. This is convenient because, otherwise, you would be responsible for adding the file extension yourself. This is obviously not always possible, nor is it practical. Using the Custom Handler file type helps you avoid any extra configuration.

Notice the class stub that the file type automatically creates for you. Listing 27-7 shows the class.

Listing 27-7: The HttpHandler page template

VB

```
<%@ WebHandler Language="VB" Class="Handler" %>

Imports System.Web

Public Class Handler
   Implements IHttpHandler

    Public Sub ProcessRequest(ByVal context As HttpContext) _
            Implements IHttpHandler.ProcessRequest
        context.Response.ContentType = "text/plain"
        context.Response.Write("Hello World")
    End Sub

    Public ReadOnly Property IsReusable() As Boolean _
            Implements IHttpHandler.IsReusable
        Get
            Return False
        End Get
    End Property

End Class
```

C#

```
<%@ WebHandler Language="C#" Class="Handler" %>

using System.Web;

public class Handler : IHttpHandler {

    public void ProcessRequest (HttpContext context) {
        context.Response.ContentType = "text/plain";
        context.Response.Write("Hello World");
    }

    public bool IsReusable {
        get {
            return false;
        }
    }

}
```

Notice that the stub implements the IHttpHandler interface, which requires the `ProcessRequest` method and `IsReusable` property. The `ProcessRequest` method is the method we use to actually process the incoming HTTP request. By default, the class stub changes the content type to plain and then writes the `"Hello World"` string to the output stream. The `IsReusable` property simply lets ASP.NET know if incoming HTTP requests can reuse the sample instance of this HttpHandler.

By default, this handler is ready to run right away. Try executing the handler in your browser and see what happens. The interesting thing to note about this handler is that because it changes the content to text/plain, browsers handle the responses from this handler in potentially very different ways depending on a number of factors:

- Browser type and version
- Applications loaded on the system that may map to the MIME type
- Operating system and service pack level

Based on these factors, you might see the text returned in the browser, you might see Notepad open and display the text, or you might receive the Open/Save/Cancel prompt from IE. Make sure you understand the potential consequences of changing the ContentType header.

You can continue the example by modifying it to return an actual file. In this case, you use the handler to return an image. To do this, you simply modify the code in the `ProcessRequest` method, as shown in Listing 27-8.

Listing 27-8: Outputting an image from an HttpHandler

VB

```
<%@ WebHandler Language="VB" Class="Handler" %>

Imports System.Web

Public Class Handler : Implements IHttpHandler

    Public Sub ProcessRequest(ByVal context As HttpContext) _
            Implements IHttpHandler.ProcessRequest
        'Logic to retrieve the image file
        context.Response.ContentType = "image/jpeg"
        context.Response.WriteFile("Garden.jpg")
    End Sub

    Public ReadOnly Property IsReusable() As Boolean _
            Implements IHttpHandler.IsReusable
        Get
            Return False
        End Get
    End Property

End Class
```

C#

```
<%@ WebHandler Language="C#" Class="Handler" %>
```

Continued

```
using System.Web;

public class Handler : IHttpHandler {

    public void ProcessRequest (HttpContext context) {
        //Logic to retrieve the image file
        context.Response.ContentType = "image/jpeg";
        context.Response.WriteFile("Garden.jpg");
    }

    public bool IsReusable {
        get {
            return false;
        }
    }

}
```

As you can see, you simply change the ContentType to image/jpeg to indicate that you are returning a JPEG image; then you use the `WriteFile()` method to write an image file to the output stream. Load the handler into a browser, and you see that the handler displays the image. Figure 27-8 shows the resulting Web page.

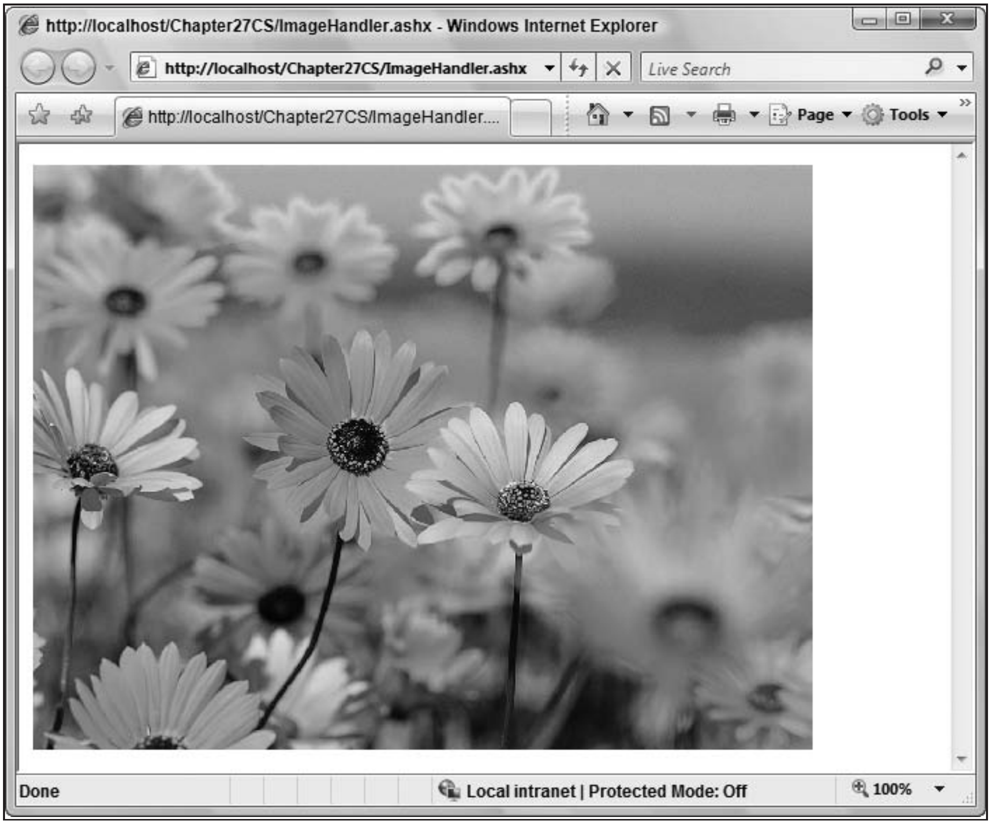

http://localhost/Chapter27CS/ImageHandler.ashx - Windows Internet Explorer
http://localhost/Chapter27CS/ImageHandler.ashx
Live Search
http://localhost/Chapter27CS/ImageHandler....
Page
Tools
Done
Local intranet | Protected Mode: Off
100%

Figure 27-8

Now, you create a simple Web page to display the image handler. Listing 27-9 shows code for the Web page.

Listing 27-9: A sample Web page using the HttpHandler for the image source

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/
xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>HttpHandler Serving an Image</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <img src="Handler.ashx" />
    </div>
    </form>
</body>
</html>
```

Although this sample is simple, you can enhance it by passing querystring parameters to your handler and using them to perform additional logic in the handler. For instance, you can pass an ID in to dynamically retrieve an image from a SQL database and return it to the client, like this:

```
<img src="Handler.ashx?imageid=123" />
```

Mapping a File Extension in IIS

Although using the `.ashx` file extension is convenient, you might want to create an HTTP handler for a custom file extension or even for a commonly used extension. Use the code from the image handler to demonstrate this.

Create a new class in the `App_Code` directory of your Web project. You can simply copy the code from the existing image handler control into this class, as shown in Listing 27-10. Notice that you removed the `WebHandler` directive because this is only a class and not a generic handler control. Other than that, the code is the same.

Listing 27-10: The class-based image HttpHandler

VB

```
Imports System.Web

Public Class MappedHandler : Implements IHttpHandler

    Public Sub ProcessRequest(ByVal context As HttpContext) _
            Implements IHttpHandler.ProcessRequest
        context.Response.ContentType = "image/jpeg"
        context.Response.WriteFile("Garden.jpg")
    End Sub

    Public ReadOnly Property IsReusable() As Boolean _
            Implements IHttpHandler.IsReusable
```

Continued

```
        Get
            Return False
        End Get
    End Property

End Class
```

C#

```
using System.Web;

public class MappedHandler : IHttpHandler {

    public void ProcessRequest (HttpContext context) {
        //Logic to retrieve the image file
        context.Response.ContentType = "image/jpeg";
        context.Response.WriteFile("Garden.jpg");
    }

    public bool IsReusable {
        get {
            return false;
        }
    }

}
```

After your class is added, configure the application to show which file extension this handler serves. You do this by adding an httpHandlers section to `web.config`. Listing 27-11 shows the section to add for the image handler.

Listing 27-11: Adding the HttpHandler configuration information to web.config

```
<httpHandlers>
     <add verb="*" path="ImageHandler.img" type="MappedHandler, App_Code" />
</httpHandlers>
```

In the configuration section, you direct the application to use the `MappedHandler` class to process incoming requests for `ImageHandler.img`. You can also specify wildcards for the path. Specifying `*.img` for the path indicates that you want the application to use the `MappedHandler` class to process any request with the `.img` file extension. Specifying `*` indicates that you want all requests to the application to be processed using the handler.

As with HttpModules, if you are running your Web application using IIS 7, then you will also need to add the HttpHandler configuration section to the `<system.webServer>` configuration section of your applications config file. When adding the handler configuration in this section, you also need to include the name attribute.

```
<add name="ImageHandler" verb="*" path="ImageHandler.img"
    type="MappedHandler, App_Code" />
```

Load the `ImageHandler.img` file into a browser and, again, you should see that it serves up the image. Figure 27-9 shows the results. Notice the path in the browser's address bar leads directly to the `ImageHandler.img` file.

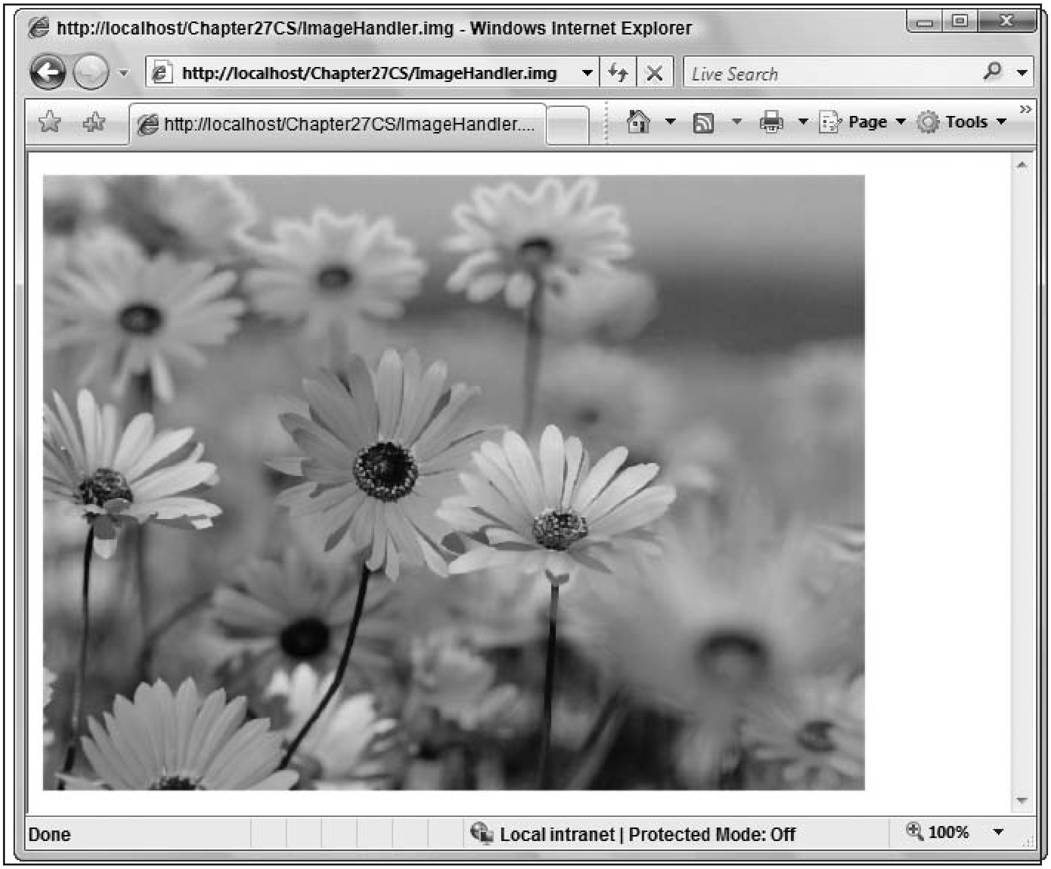

http://localhost/Chapter27CS/ImageHandler.img - Windows Internet Explorer
http://localhost/Chapter27CS/ImageHandler.img
Live Search
http://localhost/Chapter27CS/ImageHandler....
Page
Tools
Done
Local intranet | Protected Mode: Off
100%

Figure 27-9

Summary

In this chapter, you learned a number of ways you can create modules which allow you to interact with the ASP.NET request processing pipeline. First you worked with HttpModules, which give you the power to plug yourself directly into the ASP.NET page-processing pipeline. The events provided to an HttpModule give you great power and flexibility to customize your applications.

Finally, you looked at HttpHandlers. Handlers allow you to skip the ASP.NET page-processing pipeline completely and have 100 percent control over how the framework serves up requested data. You learned how to create your own image handler and then map the handler to any file or file extension you want. Using these features of ASP.NET can help you create features in your application which exercise great control over the standard page processing which ASP.NET uses.

28

Using Business Objects

One of the best practices in programming is to separate your application into workable and separate components — also known as *business objects*. This makes your applications far easier to manage and enables you to achieve the goal of code reuse because you can share these components among different parts of the same application or between entirely separate applications.

Using business components enables you to build your ASP.NET applications using a true three-tier model where the business tier is in the middle between the presentation and data tiers. In addition, using business objects enables you to use multiple languages within your ASP.NET applications. Business objects can be developed in one programming language while the code used for the presentation logic is developed in another.

If you are moving any legacy applications or aspects of these applications to an ASP.NET environment, you might find that you need to utilize various COM components. This chapter shows you how to use both .NET and COM components in your ASP.NET pages and code.

This chapter also explains how you can mingle old ActiveX (COM) DLLs with new .NET components. So when all is said and done, you should feel somewhat relieved. You will see that you have not wasted all the effort you put into building componentized applications using the ‘‘latest’’ ActiveX technologies.

Using Business Objects in ASP.NET 3.5

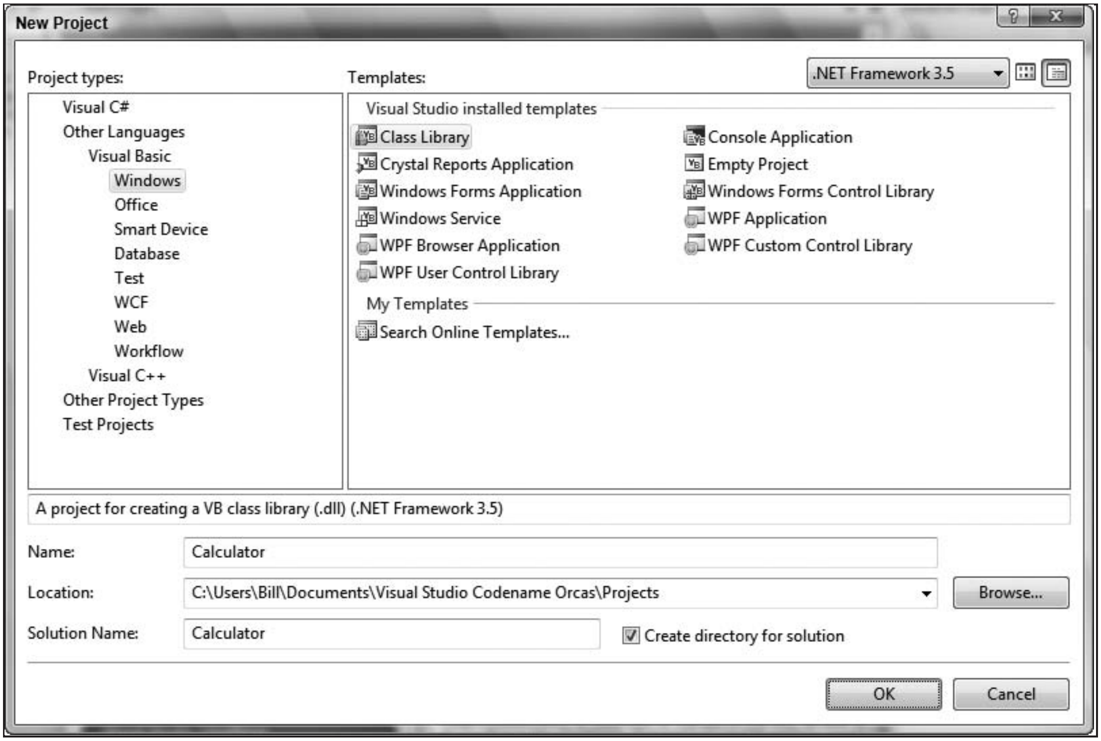
Chapter 1 of this book provides an introduction to using .NET business objects within your ASP.NET 3.5 applications. ASP.NET now includes a folder, `\App_Code`, which you can place within your ASP.NET applications to hold all your .NET business objects. The nice thing about the `App_Code` folder is that you can simply place your uncompiled .NET objects (such as `Calculator.vb` or `Calculator.cs`) into this folder and ASP.NET takes care of compiling the objects into usable .NET business objects.

Chapter 1 also shows how you can place within the `App_Code` folder multiple custom folders that enable you to use business objects written in different programming languages. Using this method enables ASP.NET to compile each business object into the appropriate DLLs to be used by your ASP.NET applications.

Creating Precompiled .NET Business Objects

Even though the `App_Code` folder is there for your use, you might choose instead to precompile your business objects into DLLs to be used by your ASP.NET applications. This is the method that was utilized prior to ASP.NET 2.0 and is still a method that is available today. You also might not have a choice if you are receiving your .NET business objects only as DLLs.

First look at how to create a simple .NET business object using Visual Studio 2008. The first step is not to create an ASP.NET project but to choose File ➪ New ➪ Project from the Visual Studio menu. This launches the New Project dialog. From this dialog, select Class Library as the project type and name the project `Calculator` (see Figure 28-1).


New Project
Project types:
Visual C#
Other Languages
Visual Basic
Windows
Office
Smart Device
Database
Test
WCF
Web
Workflow
Visual C++
Other Project Types
Test Projects
Templates:
.NET Framework 3.5
Visual Studio installed templates
Class Library
Crystal Reports Application
Windows Forms Application
Windows Service
WPF Browser Application
WPF User Control Library
Console Application
Empty Project
Windows Forms Control Library
WPF Application
WPF Custom Control Library
My Templates
Search Online Templates...
Browse...
OK
Cancel

A project for creating a VB class library (.dll) (.NET Framework 3.5)

Name: Calculator

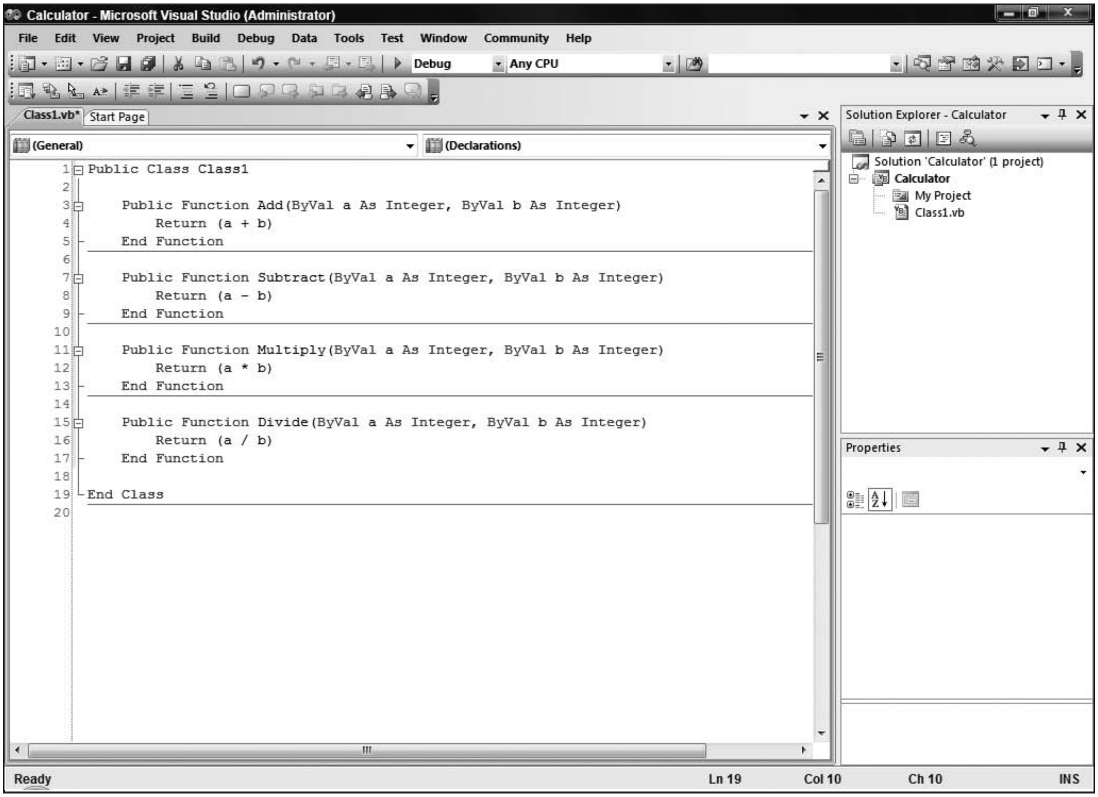
Location: C:\Users\Bill\Documents\Visual Studio Codename Orcas\Projects

Solution Name: Calculator

☑ Create directory for solution

Figure 28-1

Using the `Class1.vb` or `Class1.cs` file that is created in the project for you, modify the class to be a simple calculator with `Add`, `Subtract`, `Multiply`, and `Divide` functions. This is illustrated using Visual Basic in Figure 28-2.


Calculator - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Data Tools Test Window Community Help
Debug
Any CPU
Class1.vb*
Start Page
(General)
(Declarations)
Solution Explorer - Calculator
Solution 'Calculator' (1 project)
Calculator
My Project
Class1.vb
Properties
Ready
Ln 19
Col 10
Ch 10
INS

```
Public Class Class1

    Public Function Add(ByVal a As Integer, ByVal b As Integer)
        Return (a + b)
    End Function

    Public Function Subtract(ByVal a As Integer, ByVal b As Integer)
        Return (a - b)
    End Function

    Public Function Multiply(ByVal a As Integer, ByVal b As Integer)
        Return (a * b)
    End Function

    Public Function Divide(ByVal a As Integer, ByVal b As Integer)
        Return (a / b)
    End Function

End Class
```

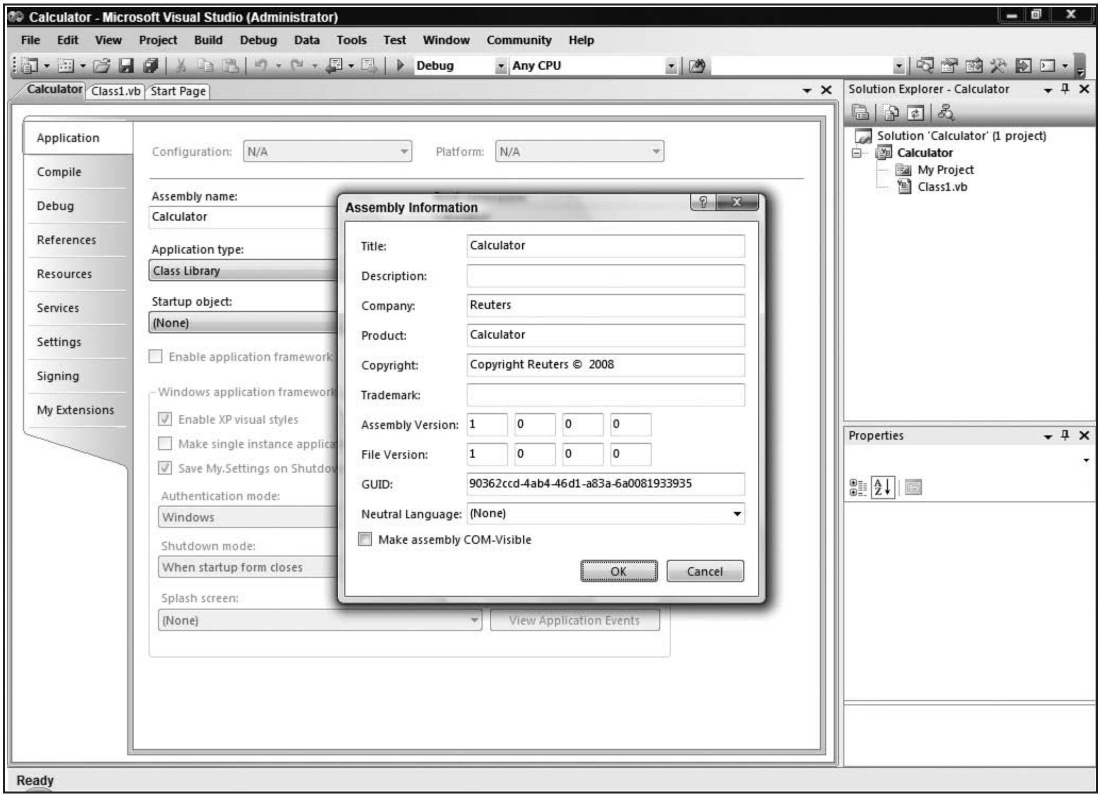
Figure 28-2

One point to pay attention to when you build your .NET components is the assembly's metadata that is stored along with the assembly. Looking at the project's properties, click the Application tab (the first tab available). Note that you can get to the project's properties by right-clicking on the project title in the Solution Explorer. On this tab's page, you will find a button labeled Assembly Information. Clicking this button gives you a dialog where you can put in the entire business object's metadata, including the assembly's versioning information (see Figure 28-3).

You are now ready to compile the business object into a usable object. To accomplish this task, choose Build ➪ Build Calculator from the Visual Studio menu. This process compiles everything contained in this solution down to a `Calculator.dll` file. You will find this DLL in your project's `bin\debug` folder. By default, that will be `C:\Users\[user]\Documents\Visual Studio 2008\Projects\Calculator\Calculator\bin\Debug\Calculator.dll`, only if you are using Windows Vista.

Besides using Visual Studio 2008 to build and compile your business objects into DLLs, you can also accomplish this yourself manually. In Notepad, you simply create the same class file as was shown in Figure 28-2 and save the file as `Calculator.vb` or `Calculator.cs` depending on the language you are using. After saving the file, you need to compile the class into an assembly (a DLL).

The .NET Framework provides you with a compiler for each of the targeted languages. This book focuses on the Visual Basic 2008 and C# 2008 compilers that come with the Framework.


Calculator - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Data Tools Test Window Community Help
Debug
Any CPU
Calculator
Class1.vb
Start Page
Application
Compile
Debug
References
Resources
Services
Settings
Signing
My Extensions
Solution Explorer - Calculator
Solution 'Calculator' (1 project)
Calculator
My Project
Class1.vb
Properties
View Application Events
OK
Cancel
Ready

Configuration: N/A
Platform: N/A
Assembly name: Calculator
Application type: Class Library
Startup object: (None)
☐ Enable application framework
Windows application framework
☑ Enable XP visual styles
☐ Make single instance applica
☑ Save My.Settings on Shutdo
Authentication mode: Windows
Shutdown mode: When startup form closes
Splash screen: (None)

Assembly Information

Title: Calculator
Description:
Company: Reuters
Product: Calculator
Copyright: Copyright Reuters © 2008
Trademark:
Assembly Version: 1 0 0 0
File Version: 1 0 0 0
GUID: 90362ccd-4ab4-46d1-a83a-6a0081933935
Neutral Language: (None)
☐ Make assembly COM-Visible

Figure 28-3

To compile this class, open the Visual Studio 2008 Command Prompt found at All Programs ➪ Microsoft Visual Studio 2008 ➪ Visual Studio Tools ➪ Visual Studio 2008 Command Prompt. From the provided DOS prompt, navigate to the directory that is holding your `Calculator` class (an example navigation command is `cd c:\My Files`). From the DOS prompt, type the following command if you are using the Visual Basic compiler:

```
vbc /t: library Calculator.vb
```

If your class is in C#, you use the following command:

```
csc /t:library Calculator.cs
```

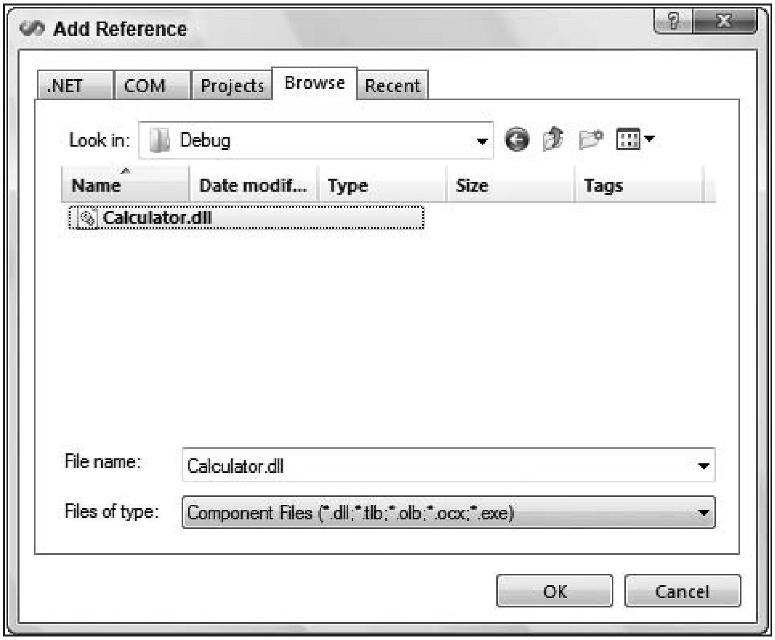
As stated, each language uses its own compiler. Visual Basic uses the `vbc.exe` compiler found at `C:\Windows\Microsoft.NET\Framework\v3.5\`. You will find the C# compiler, `csc.exe`, contained in the same folder. In the preceding examples, `/t:library` states that you are interested in compiling the `Calculator.vb` (or `.cs`) class file into a DLL and not an executable (`.exe`), which is the default. Following the `t:/library` command is the name of the file to be compiled.

There are many different commands you can give the compiler — even more than Visual Studio 2008 offers. For example, if you want to make references to specific DLLs in your assembly, you will have to *add commands such as* `/r:system.data.dll`*. To get a full list of all the compiler options, check out the MSDN documentation.*

After you have run the commands through the compiler, the DLL is created and ready to go.

Using Precompiled Business Objects in Your ASP.NET Applications

To use any DLLs in your ASP.NET 3.5 project, you need to create a `Bin` folder in the root directory of your application by right-clicking on the project within the Solution Explorer and selecting Add ASP.NET Folder ➪ Bin. In Visual Studio 2008, the `Bin` directory's icon appears as a gray folder with a gear next to it. Add your new DLL to this folder by right-clicking on the folder and selecting the Add Reference option from the menu provided. This launches the Add Reference dialog. From this dialog, select the Browse tab and browse until you find the `Calculator.dll`. When you find it, highlight the DLL and press OK to add it to the `Bin` folder of your project. This dialog is illustrated in Figure 28-4.



Look in: Debug

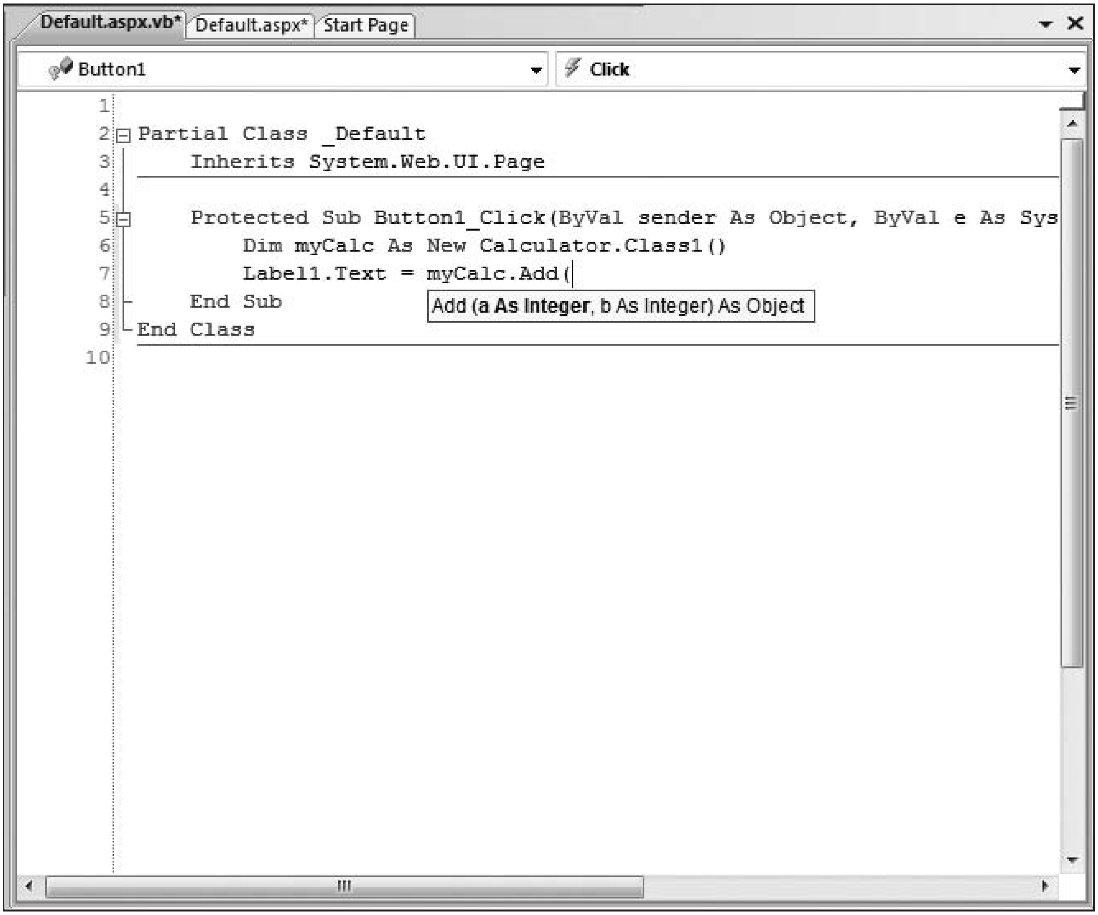
File name: Calculator.dll

Files of type: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe)

Figure 28-4

`Calculator.dll` is added to your project and is now accessible by the entire project. This means that you now have access to all the functions exposed through this interface. Figure 28-5 shows an example of how IntelliSense makes exploring this .NET component easier than ever.

As you can see, it is rather simple to create .NET components and use them in your ASP.NET applications. Next, let's look at using COM components.



```

Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As Sys
        Dim myCalc As New Calculator.Class1()
        Label1.Text = myCalc.Add(
    End Sub
End Class

```

Figure 28-5

COM Interop: Using COM Within .NET

Microsoft knows that every one of its legions of developers out there would be quite disappointed if they couldn't use the thousands of COM controls that it has built, maintained, and improved over the years. Microsoft knows that nobody would get up and walk away from these controls to a purely .NET world.

To this end, Microsoft has provided us with COM Interoperability. COM Interop (for short) is a technology that enables .NET to wrap the functionality of a COM object with the interface of a .NET component so that your .NET code can communicate with the COM object without having to use COM techniques and interfaces in your code.

Figure 28-6 illustrates the Runtime Callable Wrapper, the middle component that directs traffic between the .NET code and the COM component.

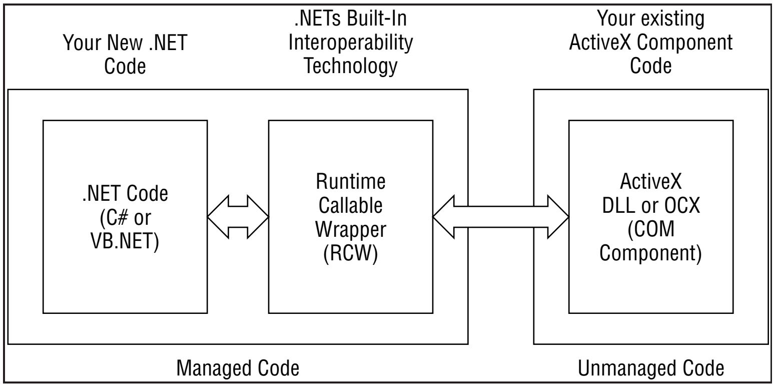

Your New .NET Code
.NETs Built-In Interoperability Technology
Your existing ActiveX Component Code
.NET Code (C# or VB.NET)
Runtime Callable Wrapper (RCW)
ActiveX DLL or OCX (COM Component)
Managed Code
Unmanaged Code

Figure 28-6

The Runtime Callable Wrapper

The Runtime Callable Wrapper, or RCW, is the magic piece of code that enables interaction to occur between .NET and COM. One RCW is created for each COM component in your project. To create an RCW for a COM component, you can use Visual Studio 2008.

To add an ActiveX DLL to the References section of your project, choose Website ➪ Add Reference or choose the Add Reference menu item that appears when you right-click the root node of your project in the Solution Explorer.

The Add Reference dialog box appears with five tabs: .NET, COM, Projects, Browse, and Recent, as shown in Figure 28-7. For this example, select the COM tab and locate the component that you want to add to your .NET project. After you have located the component, highlight the item and click OK to add a reference to the component to your project. The newly added component will then be found inside a newly created `Bin` folder in your project.

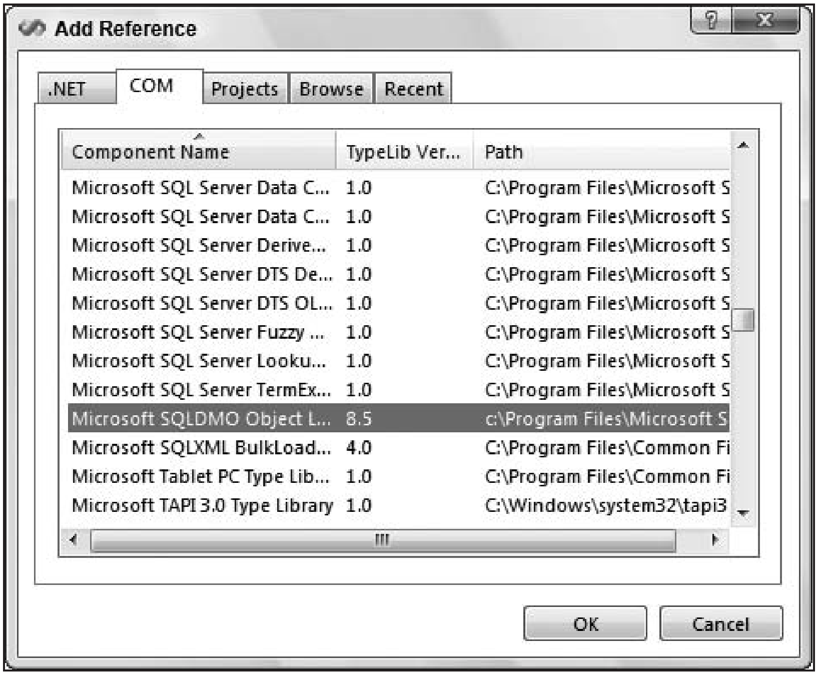
Your Interop library is automatically created for you from the ActiveX DLL that you told Visual Studio 2008 to use. This Interop library is the RCW component customized for your ActiveX control, as shown previously in Figure 28-6. The name of the Interop file is simply `Interop.OriginalName.DLL`.

It is also possible to create the RCW files manually instead of doing it through Visual Studio 2008. In the .NET Framework, you will find a method to create RCW Interop files for controls manually through a command-line tool called the Type Library Importer. You invoke the Type Library Importer by using the `tlbimp.exe` executable.

For example, to create the Interop library for the SQLDMO object used earlier, start up a Visual Studio 2008 Command Prompt from the Microsoft Visual Studio 2008 ➪ Visual Studio Tools group within your Start Menu. From the comment prompt, type

```
tlbimp sqldmo.dll /out:sqldmoex.dll
```

In this example, the `/out:` parameter specifies the name of the RCW Interop library to be created. If you omit this parameter, you get the same name that Visual Studio would generate for you.



Component Name	TypeLib Ver...	Path
Microsoft SQL Server Data C...	1.0	C:\Program Files\Microsoft S
Microsoft SQL Server Data C...	1.0	C:\Program Files\Microsoft S
Microsoft SQL Server Derive...	1.0	C:\Program Files\Microsoft S
Microsoft SQL Server DTS De...	1.0	C:\Program Files\Microsoft S
Microsoft SQL Server DTS OL...	1.0	C:\Program Files\Microsoft S
Microsoft SQL Server Fuzzy ...	1.0	C:\Program Files\Microsoft S
Microsoft SQL Server Looku...	1.0	C:\Program Files\Microsoft S
Microsoft SQL Server TermEx...	1.0	C:\Program Files\Microsoft S
Microsoft SQLDMO Object L...	8.5	c:\Program Files\Microsoft S
Microsoft SQLXML BulkLoad...	4.0	C:\Program Files\Common Fi
Microsoft Tablet PC Type Lib...	1.0	C:\Program Files\Common Fi
Microsoft TAPI 3.0 Type Library	1.0	C:\Windows\system32\tapi3

Figure 28-7

The Type Library Importer is useful when you are not using Visual Studio 2008 as your development environment, if you want to have more control over the assemblies that get created for you, or if you are automating the process of connecting to COM components.

The Type Library Importer is a wrapper application around the `TypeLibConvertor` class of the `System.Runtime.InteropServices` namespace.

Using COM Objects in ASP.NET Code

To continue working through some additional examples, you next take a look at a simple example of using a COM object written in Visual Basic 6 within an ASP.NET page.

In the first step, you create an ActiveX DLL that you can use for the upcoming examples. Add the Visual Basic 6 code shown in Listing 28-1 to a class called `NameFunctionsClass` and compile it as an ActiveX DLL called `NameComponent.dll`.

Listing 28-1: VB6 code for ActiveX DLL, NameComponent.DLL

```
Option Explicit

Private m_sFirstName As String
Private m_sLastName As String

Public Property Let FirstName(Value As String)
  m_sFirstName = Value
```

```
End Property

Public Property Get FirstName() As String
  FirstName = m_sFirstName
End Property

Public Property Let LastName(Value As String)
  m_sLastName = Value
End Property

Public Property Get LastName() As String
  LastName = m_sLastName
End Property

Public Property Let FullName(Value As String)
  m_sFirstName = Split(Value, " ")(0)
  If (InStr(Value, " ") > 0) Then
    m_sLastName = Split(Value, " ")(1)
  Else
    m_sLastName = ""
  End If
End Property

Public Property Get FullName() As String
  FullName = m_sFirstName + " " + m_sLastName
End Property

Public Property Get FullNameLength() As Long
  FullNameLength = Len(Me.FullName)
End Property
```

Now that you have created an ActiveX DLL to use in your ASP.NET pages, the next step is to create a new ASP.NET project using Visual Studio 2008. Replace the HTML code in the `Default.aspx` file with the HTML code illustrated in Listing 28-2. This adds a number of text boxes and labels to the HTML page, as well as the Visual Basic or C# code for the functionality.

Listing 28-2: Using NameComponent.dll

VB

```
<%@ Page Language="VB" %>

<script runat="server">
  Protected Sub AnalyzeName_Click(ByVal sender As Object, _
     ByVal e As System.EventArgs)

    Dim Name As New NameComponent.NameFunctionsClass()

    If (FirstName.Text.Length > 0) Then
      Name.FirstName = FirstName.Text
    End If

    If (LastName.Text.Length > 0) Then
```

Continued

```
      Name.LastName = LastName.Text
    End If

    If (FullName.Text.Length > 0) Then
      Name.FullName = FullName.Text
    End If

    FirstName.Text = Name.FirstName
    LastName.Text = Name.LastName
    FullName.Text = Name.FullName
    FullNameLength.Text = Name.FullNameLength.ToString

  End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
     <title>Using COM Components</title>
  </head>
  <body>
    <form id="form1" runat="server">
      <p>
        <asp:Label ID="Label1" runat="server">First Name:</asp:Label>
        &nbsp;
        <asp:TextBox ID="FirstName" runat="server"></asp:TextBox>
      </p>
      <p>
        <asp:Label ID="Label2" runat="server">Last Name:</asp:Label>
        &nbsp;
        <asp:TextBox ID="LastName" runat="server"></asp:TextBox>
      </p>
      <p>
        <asp:Label ID="Label3" runat="server">Full Name:</asp:Label>
        &nbsp;
        <asp:TextBox ID="FullName" runat="server"></asp:TextBox>
      </p>
      <p>
        <asp:Label ID="Label4" runat="server">Full Name Length:</asp:Label>
        &nbsp;
        <asp:Label ID="FullNameLength" runat="server"
         Font-Bold="True">0</asp:Label>
      </p>
      <p>
        <asp:Button ID="AnalyzeName" runat="server"
         OnClick="AnalyzeName_Click" Text="Analyze Name"></asp:Button>
      </p>
    </form>
  </body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
```

```
    protected void AnalyzeName_Click(object sender, System.EventArgs e)
    {
      NameComponent.NameFunctionsClass Name =
         new NameComponent.NameFunctionsClass();

      if (FirstName.Text.Length > 0)
      {
        string firstName = FirstName.Text.ToString();
        Name.set_FirstName(ref firstName);
      }

      if (LastName.Text.Length > 0)
      {
        string lastName = LastName.Text.ToString();
        Name.set_LastName(ref lastName);
      }

      if (FullName.Text.Length > 0)
      {
        string fullName = FullName.Text.ToString();
        Name.set_FullName(ref fullName);
      }

      FirstName.Text = Name.get_FirstName();
      LastName.Text = Name.get_LastName();
      FullName.Text = Name.get_FullName();
      FullNameLength.Text = Name.FullNameLength.ToString();
    }
  </script>
```

Now you need to add the reference to the ActiveX DLL that you created in the previous step. To do so, follow these steps:

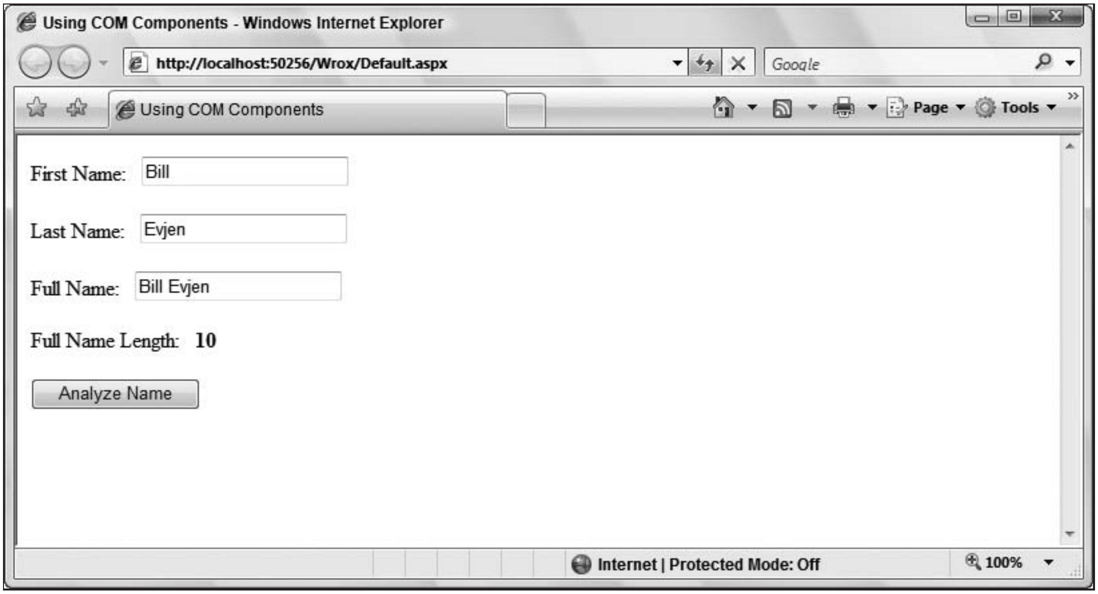
1. Right-click your project in the Solution Explorer dialog.
2. Select the Add Reference menu item.
3. In the Add Reference dialog box, select the fourth tab, Browse.
4. Locate the `NameComponent.dll` object by browsing to its location.
5. Click OK to add `NameComponent.dll` to the list of selected components and close the dialog box.

> *If you are not using Visual Studio 2008 or code-behind pages, you can still add a reference to your COM control by creating the RCW manually using the Type Library Converter and then placing an* `Imports` *statement (VB) or* `using` *statement (C#) in the page.*

After you have selected your component using the Add Reference dialog, an RCW file is created for the component and added to your application.

That's all there is to it! Simply run the application to see the COM interoperability layer in action.

Figure 28-8 shows the ASP.NET page that you created. When the Analyze Name button is clicked, the fields in the First Name, Last Name, and Full Name text boxes are sent to the RCW to be passed to the `NameComponent.DLL` ActiveX component. Data is retrieved in the same manner to repopulate the text boxes and to indicate the length of the full name.


Using COM Components - Windows Internet Explorer
http://localhost:50256/Wrox/Default.aspx
Google
Using COM Components
Page
Tools
Internet | Protected Mode: Off
100%

First Name: Bill

Last Name: Evjen

Full Name: Bill Evjen

Full Name Length: **10**

Analyze Name

Figure 28-8

Accessing Tricky COM Members in C#

Sometimes, some members of COM objects do not expose themselves properly to C#. In the preceding examples, the `String` properties did not expose themselves, but the `Long` property (`FullNameLength`) did.

You know when there is a problem because, although you can see the property, you cannot compile the application. For instance, instead of the code shown in Listing 28-2 for C#, use the following piece of code to set the `FirstName` property of the `NameComponent.dll` ActiveX component:

```
if (FirstName.Text.Length > 0)
  Name.FirstName = FirstName.Text.ToString();
```

When you try to compile this code, you get the following error:

```
c:\inetpub\wwwroot\wrox\Default.aspx.cs(67): Property, indexer, or event 'FirstName'
is not supported by the language; try directly calling accessor methods 'NameComponent
.NameFunctionsClass.get_FirstName()' or 'NameComponent.NameFunctionsClass
.set_FirstName(ref string)'
```

The `FirstName` property seems to be fine. It shows up in IntelliSense, but you can't use it. Instead, you must use `set_FirstName` (and `get_FirstName` to read). These methods do not show up in IntelliSense, but rest assured, they exist.

Furthermore, these methods expect a `ref string` parameter rather than a `String`. In the example from Listing 28-2, two steps are used to do this properly. First, `String` is assigned to a local variable, and then the variable is passed to the method using `ref`.

Releasing COM Objects Manually

One of the great things about .NET is that it has its own garbage collection — it can clean up after itself. This is not always the case when using COM interoperability, however. .NET has no way of knowing when to release a COM object from memory because it does not have the built-in garbage collection mechanism that .NET relies on.

Because of this limitation, you should release COM objects from memory as soon as possible using the `ReleaseComObject` class of the `System.Runtime.InteropServices.Marshal` class:

C#

```
System.Runtime.InteropServices.Marshal.ReleaseComObject(Object);
```

It should be noted that if you attempt to use this object again before it goes out of scope, you would raise an exception.

Error Handling

Error handling in .NET uses exceptions instead of the HRESULT values used by Visual Basic 6 applications. Luckily, the RCW does most of the work to convert between the two.

Take, for instance, the code shown in Listing 28-3. In this example, a user-defined error is raised if the numerator or the denominator is greater than 1000. Also notice that we are not capturing a divide-by-zero error. Notice what happens when the ActiveX component raises the error on its own.

Begin this example by compiling the code listed in Listing 28-3 into a class named `DivideClass` within an ActiveX component called `DivideComponent.dll`.

Listing 28-3: Raising errors in VB6

```
Public Function DivideNumber(Numerator As Double, _
                             Denominator As Double) As Double

  If ((Numerator > 1000) Or (Denominator > 1000)) Then
      Err.Raise vbObjectError + 1, _
                "DivideComponent:Divide.DivideNumber", _
                "Numerator and denominator both have to " + _
                "be less than or equal to 1000."

  End If

  DivideNumber = Numerator / Denominator

End Function
```

Next, create a new ASP.NET project; add a reference to the `DivideComponent.dll` (invoking Visual Studio 2008 to create its own copy of the RCW). Remember, you can also do this manually by using the `tlbimp` executable.

Now add the code shown in Listing 28-4 to an ASP.NET page.

Listing 28-4: Error handling in .NET

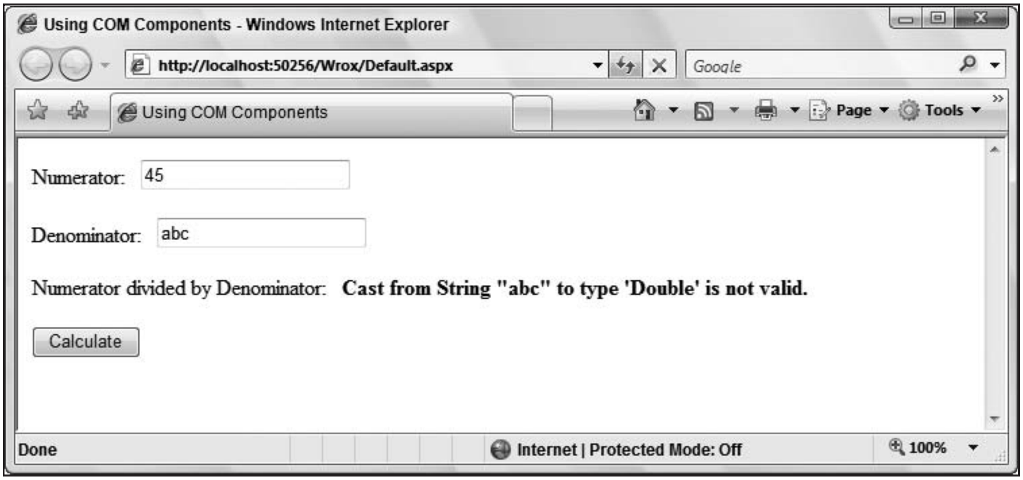
VB

```
<%@ Page Language="VB" %>

<script runat="server">
  Protected Sub Calculate_Click(ByVal sender As Object, _
     ByVal e As System.EventArgs)

    Dim Divide As New DivideComponent.DivideClass()

    Try
       Answer.Text = Divide.DivideNumber(Numerator.Text, Denominator.Text)
    Catch ex As Exception
       Answer.Text = ex.Message.ToString()
    End Try

    System.Runtime.InteropServices.Marshal.ReleaseComObject(Divide)

  End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
     <title>Using COM Components</title>
  </head>
  <body>
    <form id="form1" runat="server">
      <p>
        <asp:Label ID="Label1" runat="server">Numerator:</asp:Label>
        &nbsp;
        <asp:TextBox ID="Numerator" runat="server"></asp:TextBox>
      </p>
      <p>
        <asp:Label ID="Label2" runat="server">Denominator:</asp:Label>
        &nbsp;
        <asp:TextBox ID="Denominator" runat="server"></asp:TextBox>
      </p>
      <p>
        <asp:Label ID="Label3" runat="server">
         Numerator divided by Denominator:</asp:Label>
        &nbsp;
        <asp:Label ID="Answer" runat="server" Font-Bold="True">0</asp:Label>
      </p>
      <p>
        <asp:Button ID="Calculate"
         runat="server"
         OnClick="Calculate_Click"
         Text="Calculate">
        </asp:Button>
      </p>
    </form>
  </body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
  protected void Calculate_Click(object sender, System.EventArgs e)
  {

    DivideComponent.DivideClass myDivide = new DivideComponent.DivideClass();

    try
    {
      double numerator = double.Parse(Numerator.Text);
      double denominator = double.Parse(Denominator.Text);
      Answer.Text = myDivide.DivideNumber(ref numerator,
         ref denominator).ToString();
    }

    catch (Exception ex)
    {
      Answer.Text = ex.Message.ToString();
    }

    System.Runtime.InteropServices.Marshal.ReleaseComObject(myDivide);

  }
</script>
```

The code in Listing 28-4 passes the user-entered values for the `Numerator` and `Denominator` to the `DivideComponent.dll` ActiveX component for it to divide. Running the application with invalid data gives the result shown in Figure 28-9.


Using COM Components - Windows Internet Explorer
http://localhost:50256/Wrox/Default.aspx
Google
Using COM Components
Page
Tools
Done
Internet | Protected Mode: Off
100%

Numerator: 45

Denominator: abc

Numerator divided by Denominator: **Cast from String "abc" to type 'Double' is not valid.**

Calculate

Figure 28-9

Depending on the language that you are using to run the ASP.NET application, you will see different values for different sets of data. For valid inputs, you will always see the correct result, of course, and for any input that is over 1000, you see the Visual Basic 6 appointed error description of `Numerator and denominator both have to be less than or equal to 1000`.

However, for invalid Strings, Visual Basic 2008 reports `Cast from string "abc" to type 'Double' is not valid.` whereas C# reports `Input string was not in a correct format.`. For a divide by zero, they both report `Divide by Zero` because the error is coming directly from the Visual Basic 6 runtime.

Deploying COM Components with .NET Applications

Deploying COM components with your .NET applications is very easy; especially when compared to just deploying ActiveX controls. Two scenarios are possible when deploying .NET applications with COM components:

- ❑ Using private assemblies
- ❑ Using shared assemblies

Private Assemblies

Installing all or parts of the ActiveX component local to the .NET application is considered installing private assemblies. In this scenario, each installation of your .NET application on the same machine has, at least, its own copy of the Interop library for the ActiveX component you are referencing, as shown in Figure 28-10.

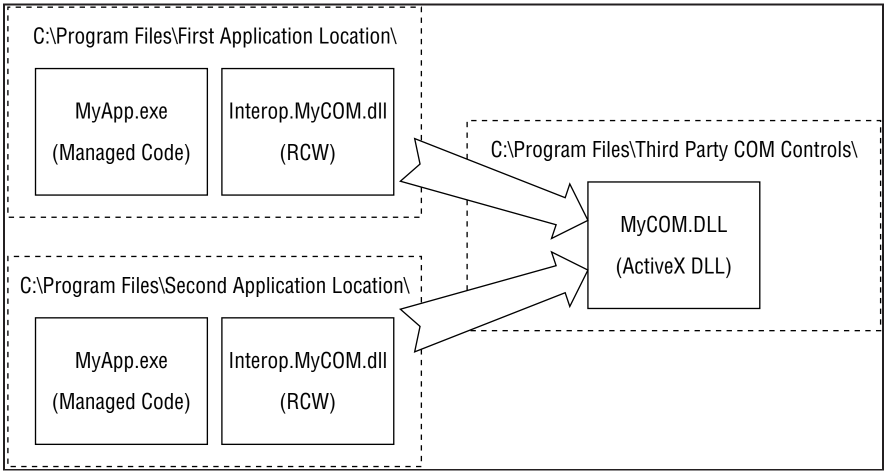

C:\Program Files\First Application Location\
MyApp.exe
(Managed Code)
Interop.MyCOM.dll
(RCW)
C:\Program Files\Third Party COM Controls\
MyCOM.DLL
(ActiveX DLL)
C:\Program Files\Second Application Location\
MyApp.exe
(Managed Code)
Interop.MyCOM.dll
(RCW)

Figure 28-10

It is up to you whether you decide to install the ActiveX component as local to the application or in a shared directory for all calling applications.

It was once considered proper practice to separate ActiveX components into their own directory so that if these components were referenced again by other applications, you did not have to register or install the *file for a second time. Using this method meant that when you upgraded a component, you automatically upgraded all the utilizing applications. However, this practice didn't work out so well. In fact, it became a very big contributor to DLL hell and the main reason why Microsoft began promoting the practice of installing private .NET component assemblies.*

After you have your components physically in place, the only remaining task is to register the ActiveX component using regsvr32, just as you would when deploying an ActiveX-enabled application.

Public Assemblies

The opposite of a private assembly is a public assembly. Public assemblies share the RCW Interop DLL for other applications. In order to create a public assembly, you must put the RCW file into the *Global Assembly Cache* (GAC), as shown in Figure 28-11.

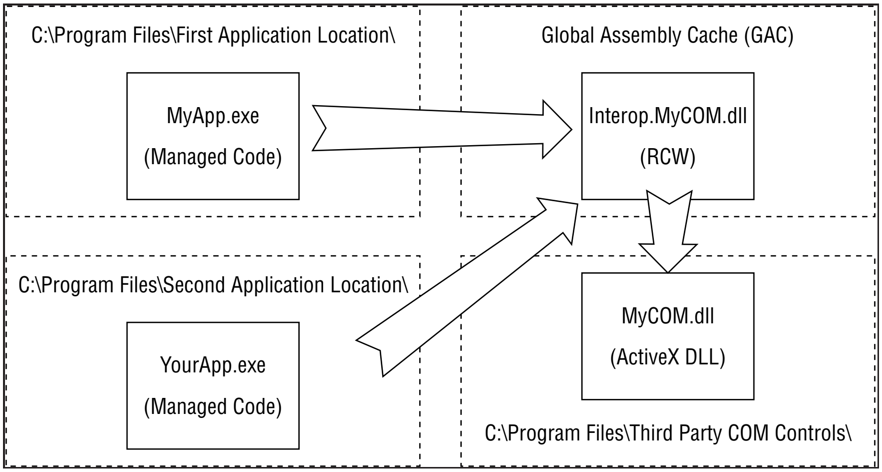

C:\Program Files\First Application Location\
Global Assembly Cache (GAC)
MyApp.exe
(Managed Code)
Interop.MyCOM.dll
(RCW)
C:\Program Files\Second Application Location\
MyCOM.dll
(ActiveX DLL)
YourApp.exe
(Managed Code)
C:\Program Files\Third Party COM Controls\

Figure 28-11

You can find the GAC at `C:\Windows\assembly`. Installing items in the GAC can be as simple as dragging-and-dropping the item into this folder through Windows Explorer. Although the GAC is open to everyone, it is not recommended that you blindly install your components into this section unless you have a very good reason to do so.

You can also add items to the GAC from the command line using the Global Assembly Cache Tool (Gacutil.exe). It enables you to view and manipulate the contents of the global assembly cache and download cache. While the Explorer view of the GAC provides similar functionality, you can use Gacutil.exe from build scripts, makefile files, and batch files.

It is hard to find a very good reason to install your ActiveX Interop Assemblies into the GAC. If we had to pick a time to do this, it would be if and when we had a highly shared ActiveX component that many .NET applications would be utilizing on the same machine. In a corporate environment, this might occur when you are upgrading existing business logic from ActiveX to .NET enablement on a server that many applications use. In a commercial setting, we avoid using the GAC.

Using .NET from Unmanaged Code

.NET provides the opposite of COM interoperability by enabling you to use your newly created .NET components within unmanaged code. This section discusses using .NET components with Visual Basic 6 executables. The techniques shown in this section are identical when you are using ActiveX OCXs or DLLs instead of executables.

The COM-Callable Wrapper (CCW) is the piece of the .NET Framework that enables unmanaged code to communicate with your .NET component. The CCW, unlike the RCW, is not a separate DLL that you distribute with your application. Instead, the CCW is part of the .NET Framework that gets instantiated once for each .NET component that you are using.

Figure 28-12 shows how the CCW marshals the communication between the unmanaged code and the .NET component in much the same way that the RCW marshals the code between managed code and COM code.

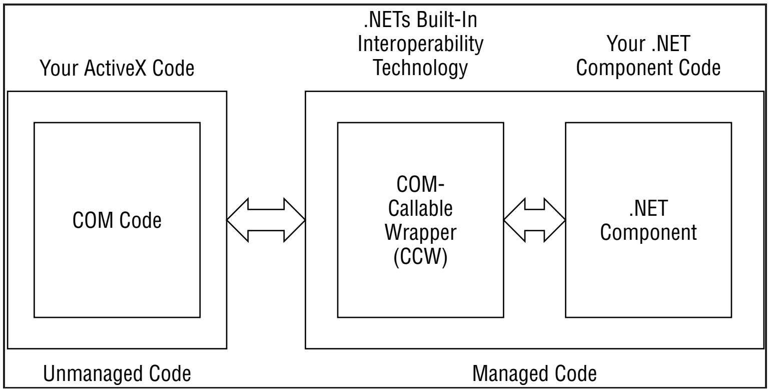


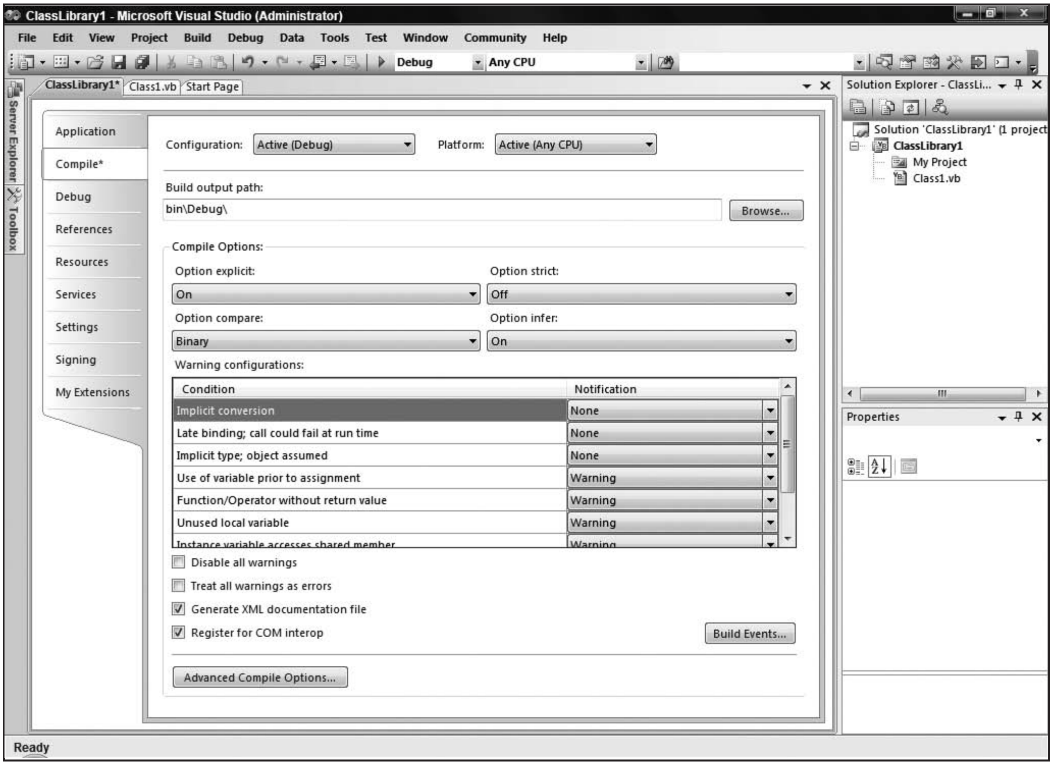
Figure 28-12

The COM-Callable Wrapper

The COM-Callable Wrapper or CCW, as previously stated, is not a separate DLL like the RCW. Instead, the CCW uses a specially created type library based on the .NET component. This type library is called an Interop Type Library. The Interop Type Library is statically linked with the unmanaged code so that this code can communicate with the CCW about the .NET component included in your application.

In order for a .NET component to generate an Interop Type Library, you tell Visual Studio 2008 to generate it when the component is built. Both Visual Basic and C# projects have a setting in the Compile properties section of the Class Library project's Property Pages dialog.

Right-click the project in the Solution Explorer and choose Properties to see the project's properties. Figure 28-13 shows the project's properties for a Visual Basic 2008 Class Library application. This is shown directly in the Visual Studio document window.


ClassLibrary1 - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Data Tools Test Window Community Help
Debug
Any CPU
ClassLibrary1* Class1.vb Start Page
Server Explorer
Toolbox
Application
Compile*
Debug
References
Resources
Services
Settings
Signing
My Extensions
Browse...
Build Events...
Advanced Compile Options...
Solution Explorer - ClassLi...
Solution 'ClassLibrary1' (1 project
ClassLibrary1
My Project
Class1.vb
Properties
Ready

Configuration: Active (Debug)
Platform: Active (Any CPU)
Build output path: bin\Debug\

Compile Options:
Option explicit: On
Option strict: Off
Option compare: Binary
Option infer: On

Warning configurations:

Condition	Notification
Implicit conversion	None
Late binding; call could fail at run time	None
Implicit type; object assumed	None
Use of variable prior to assignment	Warning
Function/Operator without return value	Warning
Unused local variable	Warning
Instance variable accesses shared member	Warning

☐ Disable all warnings
☐ Treat all warnings as errors
☑ Generate XML documentation file
☑ Register for COM interop

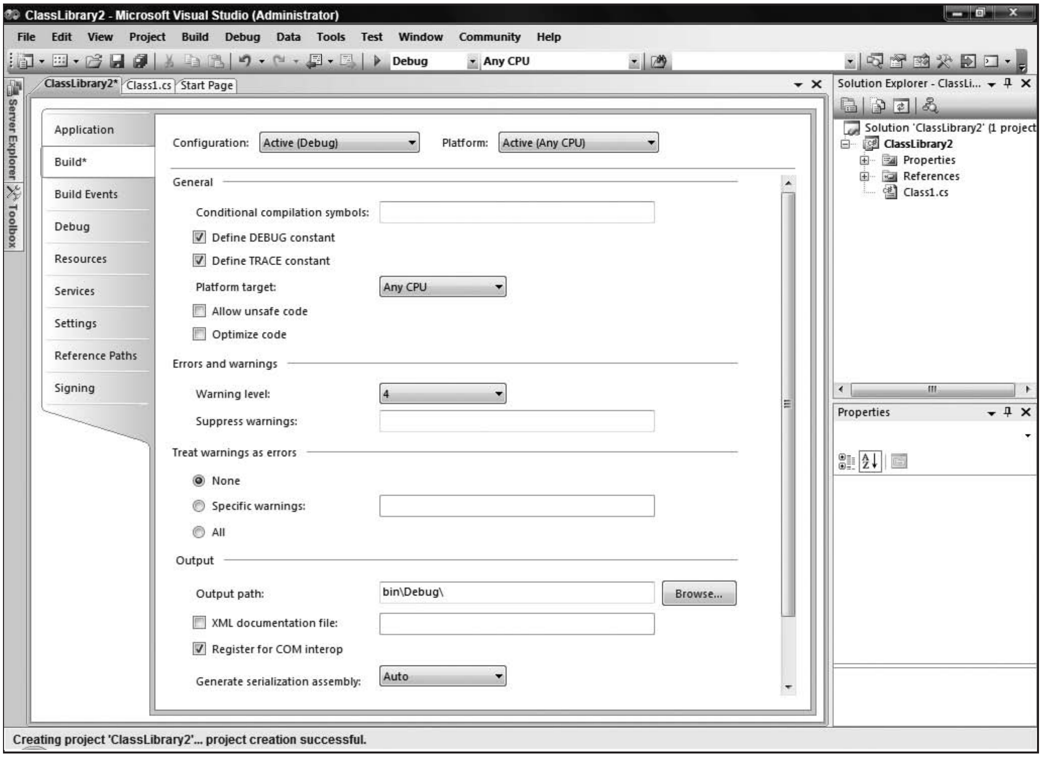
Figure 28-13

C# has a slightly different dialog, as shown in Figure 28-14. In both dialogs, the property is called Register for COM Interop. In Visual Basic, you can find this property on the Compile page; in C#, you can find it on the Build tab of the properties pages.

After you set this option by checking the check box, when you build the project a separate type library file (`.tlb`) is generated for the DLL that you are building. This `.tlb` file is your key to including .NET components in COM applications.

Normally in Visual Basic, when you add a reference to a DLL, you navigate from the References section of the Visual Basic project to find the ActiveX DLL that you want to add. If you use .NET components, they cannot be properly referenced in this manner because they are not ActiveX. Instead, you reference the Interop Type Library, which makes the functionality of the corresponding .NET component available to your application.

The .NET Framework also gives you a method to create Interop Type Library files manually for .NET components. You do this through a command-line tool called the Type Library Exporter (as compared to the Type Library Importer used for COM Interoperability). The Type Library Exporter is invoked using the `tlbexp.exe` executable.


ClassLibrary2 - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Data Tools Test Window Community Help
Debug
Any CPU
ClassLibrary2*
Class1.cs
Start Page
Server Explorer
Toolbox
Application
Build*
Build Events
Debug
Resources
Services
Settings
Reference Paths
Signing
Browse...
Solution Explorer - ClassLi...
Solution 'ClassLibrary2' (1 project
ClassLibrary2
Properties
References
Class1.cs
Properties

Configuration: Active (Debug)
Platform: Active (Any CPU)

General

Conditional compilation symbols:

☑ Define DEBUG constant

☑ Define TRACE constant

Platform target: Any CPU

☐ Allow unsafe code

☐ Optimize code

Errors and warnings

Warning level: 4

Suppress warnings:

Treat warnings as errors

● None

○ Specific warnings:

○ All

Output

Output path: bin\Debug\

☐ XML documentation file:

☑ Register for COM interop

Generate serialization assembly: Auto

Creating project 'ClassLibrary2'... project creation successful.

Figure 28-14

For example, to create the Interop Type Library for the `NameComponent.dll` in the next example, you use the following command:

```
tlbexp NameComponent.dll /out:NameComponentEx.tlb
```

The `/out:` parameter specifies the name of the Interop Type Library that is to be created. If you omit this parameter, you get a file with the same name as the ActiveX component, but with a `.tlb` extension.

The Type Library Exporter is useful when you are not using Visual Studio 2008 as your development environment, if you want to have more control over the assemblies that get created for you, or if you are automating the process of connecting to .NET components.

Using .NET Components Within COM Objects

The next example illustrates how .NET components can be utilized within COM code. To begin, create and compile the .NET code found in Listing 28-5 in either Visual Basic or C#.

After you have typed your code into your Class Library project, build the component and call it `NameComponent`. Remember to choose to include the Register for the COM Interop setting of True (by checking the appropriate check box) from the project properties pages, as shown in Figure 28-13 for Visual Basic code and Figure 28-14 for C# code. If you aren't using Visual Studio 2008, you can use `tblexp.exe` to generate the Interop Type Library manually as described previously.

Listing 28-5: The .NET component

VB

```
Public Class NameFunctions

  Private m_FirstName As String
  Private m_LastName As String

  Public Property FirstName() As String
    Get
      Return m_FirstName
    End Get

    Set(ByVal Value As String)
      m_FirstName = Value
    End Set
  End Property

  Public Property LastName() As String
    Get
      Return m_LastName
    End Get

    Set(ByVal Value As String)
      m_LastName = Value
    End Set
  End Property

  Public Property FullName() As String
    Get
      Return m_FirstName + " " + m_LastName
    End Get

    Set(ByVal Value As String)
      m_FirstName = Split(Value, " ")(0)
      m_LastName = Split(Value, " ")(1)
    End Set
  End Property

  Public ReadOnly Property FullNameLength() As Long
    Get
      FullNameLength = Len(Me.FullName)
    End Get
  End Property

End Class
```

C#

```
using System;
using System.Runtime.InteropServices;

namespace NameComponent
```

Continued

```
{
  [ComVisible(true)]
  public class NameFunctions
  {

    private string m_FirstName;
    private string m_LastName;

    public string FirstName
    {
      get
      {
        return m_FirstName;
      }
      set
      {
        m_FirstName=value;
      }
    }

    public string LastName
    {
      get
      {
        return m_LastName;
      }
      set
      {
        m_LastName=value;
      }
    }

    public string FullName
    {
      get
      {
        return m_FirstName + " " + m_LastName;
      }
      set
      {
        m_FirstName=value.Split(' ')[0];
        m_LastName=value.Split(' ')[1];
      }
    }

    public long FullNameLength
    {
      get
      {
        return this.FullName.Length;
      }
    }

  }
}
```

After you have created the .NET component, you can then create the consuming Visual Basic 6 code shown in Listing 28-6.

Listing 28-6: VB6 code using the .NET component

```
Option Explicit

Public Sub Main()

  Dim o As NameComponent.NameFunctions

  Set o = New NameComponent.NameFunctions

  o.FirstName = "Bill"
  o.LastName = "Evjen"

  MsgBox "Full Name is: " + o.FullName

  MsgBox "Length of Full Name is: " + CStr(o.FullNameLength)

  o.FullName = "Scott Hanselman"

  MsgBox "First Name is: " + o.FirstName

  MsgBox "Last Name is: " + o.LastName

  o.LastName = "Evjen"

  MsgBox "Full Name is: " + o.FullName

  Set o = Nothing

End Sub
```

Remember to add a reference to the .NET component. You choose Project ➪ References and select the .NET component that was created either by Visual Studio or by manually using `tlbexp.exe`.

When you run the code in Listing 28-6, you see that Visual Basic 6 does not miss a beat when communicating with the .NET component.

It is also possible to register the assemblies yourself. Earlier you learned how to manually create Interop Type Libraries with the Type Library Exporter. This tool does not register the assemblies created but instead generates only the type library.

To register the assemblies yourself, you use the Assembly Registration Tool (`regasm.exe`). This tool is similar the `regsvr32.exe` for .NET components.

To use `regasm.exe`, use a command syntax similar to the following example:

```
regasm NameComponent.dll /tlb:NameComponentEx.tlb /regfile:NameComponent.reg
```

The `/tlb:` option specifies the name of the type library, and the `/regfile:` option specifies the name of a registry file to be created that can be used later in an installation and deployment application.

Early versus Late Binding

The preceding example illustrates the use of early binding, the technique most Visual Basic 6 developers are used to. However, in some cases, it is desirable to use late binding. Performing late binding with .NET components is similar to performing late binding with ActiveX components, as shown in Listing 28-7.

Listing 28-7: Late binding with VB6

```
Option Explicit

Public Sub Main()

  Dim o As Object

  Set o = CreateObject("NameComponent.NameFunctions")

  o.FirstName = "Bill"
  o.LastName = "Evjen"

  MsgBox "Full Name is: " + o.FullName

  MsgBox "Length of Full Name is: " + CStr(o.FullNameLength)

  o.FullName = "Scott Hanselman"

  MsgBox "First Name is: " + o.FirstName

  MsgBox "Last Name is: " + o.LastName

  o.LastName = "Evjen"

  MsgBox "Full Name is: " + o.FullName

  Set o = Nothing

End Sub
```

Error Handling

Handling errors that are raised from .NET components in Visual Basic 6 is easily accomplished via the Interop functionality. Listing 28-8 shows code for both Visual Basic and C# to throw exceptions for a custom error. When the `Numerator` or the `Denominator` parameters are greater than 1000 in the `Divide` function, a custom exception is thrown up to the calling code, which is Visual Basic 6 in this example.

Notice how the divide-by-zero error possibility is not handled in this example. This is done intentionally to demonstrate how interoperability handles unhandled errors.

Listing 28-8: Raising errors

VB

```
Public Class CustomException
  Inherits Exception
  Sub New(ByVal Message As String)
    MyBase.New(Message)
  End Sub
End Class

Public Class DivideFunction

  Public Function Divide(ByVal Numerator As Double, _
                         ByVal Denominator As Double) As Double

    If ((Numerator > 1000) Or (Denominator > 1000)) Then
      Throw New CustomException("Numerator and denominator both " & _
                          "have to be less than or equal to 1000.")
    End If

    Divide = Numerator / Denominator

  End Function

End Class
```

C#

```
using System;

namespace DivideComponent
{
    public class CustomException:Exception
    {
      public CustomException(string message):base(message)
      {
      }
    }

  public class DivideFunction
  {
    public double Divide(double Numerator, double Denominator)
    {
      if ((Numerator > 1000) || (Denominator > 1000))
        throw new CustomException("Numerator and denominator " +
                     "both have to be less than or equal to 1000.");

      return Numerator / Denominator;
    }
  }
}
```

Now that you have the code for the .NET component, compile it with the Register for COM Interop flag set to True in the project's Property Pages dialog and call the component `DivideComponent`.

The consuming Visual Basic 6 code is shown in Listing 28-9. Remember to add a reference to the Interop Type Library of the `DivideComponent` generated by Visual Studio.

Listing 28-9: VB6 experiencing .NET errors

```
Option Explicit

Public Sub Main()

  Dim o As DivideComponent.DivideFunction

  Set o = New DivideComponent.DivideFunction

  MsgBox "1 divided by 3: " + CStr(o.divide(1, 3))

  MsgBox "1 divided by 0: " + CStr(o.divide(1, 0))

  MsgBox "2000 divided by 3000: " + CStr(o.divide(2000, 3000))

  Set o = Nothing

End Sub
```

The Visual Basic 6 code example in Listing 28-9 does not handle the errors thrown by the .NET component, but it can easily do so using `On Error`, Visual Basic 6's method for trapping raised errors.

Instead of trapping the errors, make sure that the Error Trapping setting in the Options dialog of Visual Basic 6 is set to Break in Class Module.

When the application is run, the first example of 1 divided by 3 works fine; you see the output properly. The second example, which you would expect to end in a divide-by-zero error, does not. Instead, an invalid property value is returned to Visual Basic 6. The final example, which does not pass the custom error handling in the .NET component, raises a Visual Basic error, as you would expect.

Deploying .NET Components with COM Applications

Deploying .NET components with COM applications is similar to deploying COM components. There are two scenarios in this deployment scheme:

- ❑ Using private assemblies
- ❑ Using shared assemblies

The following sections discuss these two scenarios.

Private Assemblies

Private assemblies mean the deployment of the .NET component is installed in each individual directory where the application is installed, within the same machine. The only needed component is the .NET DLL and the calling application. The Interop Type Library that you created earlier with Visual Studio 2008 or `tlbexp.exe` is statically linked with the component or application that references the .NET component.

The only additional task you must complete is to properly register the .NET assembly using `regasm.exe`. This is an extra step that is not needed in 100 percent .NET applications; it is required only for the interoperability for the unmanaged code to reference the managed code. Figure 28-15 illustrates using private assemblies.

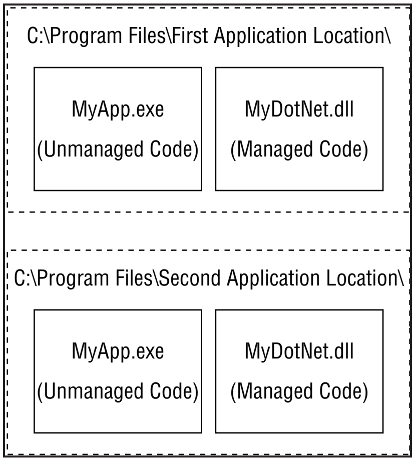

C:\Program Files\First Application Location\
MyApp.exe
(Unmanaged Code)
MyDotNet.dll
(Managed Code)
C:\Program Files\Second Application Location\
MyApp.exe
(Unmanaged Code)
MyDotNet.dll
(Managed Code)

Figure 28-15

Public Assemblies

The use of a public assembly is illustrated in Figure 28-16. This scenario involves installing the .NET component into the Global Assembly Cache (GAC).

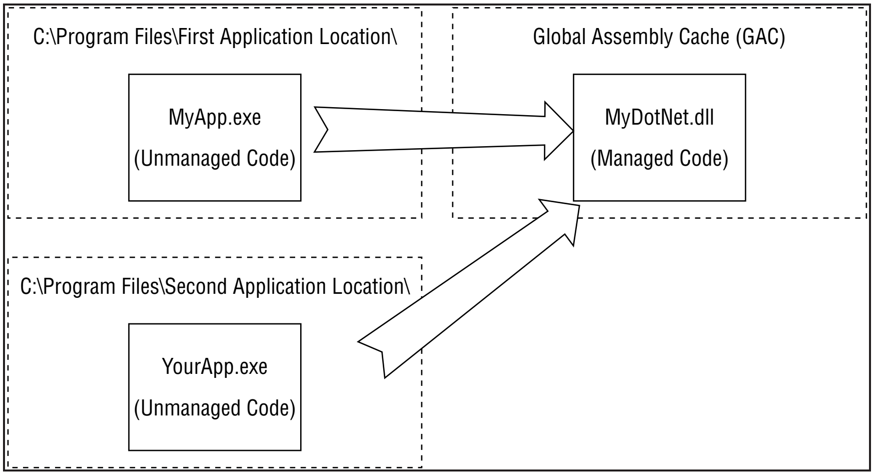

C:\Program Files\First Application Location\
MyApp.exe
(Unmanaged Code)
Global Assembly Cache (GAC)
MyDotNet.dll
(Managed Code)
C:\Program Files\Second Application Location\
YourApp.exe
(Unmanaged Code)

Figure 28-16

As with private assemblies, the .NET component and the consuming unmanaged code are the only requirements for deployment — besides the need to register the interop assembly using `regasm.exe`.

Summary

When .NET was introduced, there was some initial concern about existing ActiveX controls and their place in Microsoft's vision for the future of component development. Immediately, Microsoft stepped up to the bat and offered the robust and solid .NET Interop functionality to provide a means to communicate not only from .NET managed code to COM unmanaged code, but also from COM unmanaged code to .NET managed code. The latter was an unexpected, but welcome, feature for many Visual Basic 6 developers and future .NET component builders.

This layer of interoperability has given Microsoft the position to push .NET component development as a solution for not only newly created applications, but also applications that are currently in development and ones that have already been rolled out and are now in the maintenance phase.

Interoperability has given .NET developers a means to gradually update applications without rewriting them entirely, and it has given them a way to start new .NET projects without having to wait for all the supporting components to be developed in .NET.

29

Building and Consuming Services

When the .NET Framework 1.0 was first introduced, much of the hype around its release was focused on XML Web services. In fact, Microsoft advertised that the main purpose of the newly released .NET Framework 1.0 was to enable developers to build and consume XML Web services with ease. Unfortunately, the new Web services model was slow to be accepted by the development community because it was so radically different from those that came before. Decision makers in the development community regarded this new Web services model with a cautious eye.

Since then, Microsoft has stopped trumpeting that .NET is all about Web services and instead has really expanded the power of .NET and its relation to applications built within the enterprise. Still, the members of the IT community continued to look long and hard at the Web services model (Microsoft is no longer alone in hyping this new technology), examining how it could help them with their current issues and problems.

This chapter looks at building XML Web services and how you can consume XML Web service interfaces and integrate them into your ASP.NET applications. It begins with the foundations of XML Web services in the .NET world by examining some of the underlying technologies such as SOAP, WSDL, and more.

Communication Between Disparate Systems

It is a diverse world. In a major enterprise, very rarely do you find that the entire organization and its data repositories reside on a single vendor's platform. In most instances, organizations are made up of a patchwork of systems — some based on Unix, some on Microsoft, and some on other systems. There probably will not be a day when everything resides on a single platform where all the data moves seamlessly from one server to another. For that reason, these various systems must be able to talk to one another. If disparate systems can communicate easily, moving unique datasets around the enterprise becomes a simple process — alleviating the need for replication systems and data stores.

When XML (eXtensible Markup Language) was introduced, it became clear that the markup language would be the structure to bring the necessary integration into the enterprise. XML's power comes from the fact that it can be used regardless of the platform, language, or data store of the system using it to expose DataSets.

XML has its roots in the Standard Generalized Markup Language (SGML), which was created in 1986. Because SGML was so complex, something a bit simpler was needed — thus the birth of XML.

XML is considered ideal for data representation purposes because it enables developers to structure XML documents as they see fit. For this reason, it is also a bit chaotic. Sending self-structured XML documents between dissimilar systems does not make a lot of sense — you would have to custom build the exposure and consumption models for each communication pair.

Vendors and the industry as a whole soon realized that XML needed a specific structure that put some rules in place to clarify communication. The rules defining XML structure make the communication between the disparate systems just that much easier. Tool vendors can now automate the communication process, as well as provide for the automation of the possible creation of all the components of applications using the communication protocol.

The industry settled on using SOAP (Simple Object Access Protocol) to make the standard XML structure work. Previous attempts to solve the communication problem that arose included component technologies such as Distributed Component Object Model (DCOM), Remote Method Invocation (RMI), Common Object Request Broker Architecture (CORBA), and Internet Inter-ORB Protocol (IIOP). These first efforts failed because each of these technologies was either driven by a single vendor or (worse yet) very vendor-specific. It was, therefore, impossible to implement them across the entire industry.

SOAP enables you to expose and consume complex data structures, which can include items such as DataSets, or just tables of data that have all their relations in place. SOAP is relatively simple and easy to understand. Like ASP.NET, XML Web services are also primarily engineered to work over HTTP. The DataSets you send or consume can flow over the same Internet wires (HTTP), thereby bypassing many firewalls (as they move through port 80).

So what is actually going across the wire? ASP.NET Web services generally use SOAP over HTTP using the HTTP Post protocol. An example SOAP request (from the client to the Web service residing on a Web server) takes the structure shown in Listing 29-1.

Listing 29-1: A SOAP request

```
POST /MyWebService/Service.asmx HTTP/1.1
Host: www.wrox.com
Content-Type: text/xml; charset=utf-8
Content-Length: 19
SOAPAction: "http://tempuri.org/HelloWorld"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
```

```
    <soap:Body>
      <HelloWorld xmlns="http://tempuri.org/" />
    </soap:Body>
  </soap:Envelope>
```

The request is sent to the Web service to invoke the `HelloWorld` WebMethod (WebMethods are discussed later in this chapter). The SOAP response from the Web service is shown in Listing 29-2.

Listing 29-2: A SOAP response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 14

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <HelloWorldResponse xmlns="http://tempuri.org/">
      <HelloWorldResult>Hello World</HelloWorldResult>
    </HelloWorldResponse>
  </soap:Body>
</soap:Envelope>
```

In the examples from Listings 29-1 and 29-2, you can see that what is contained in this message is an XML file. In addition to the normal XML declaration of the `<xml>` node, you see a structure of XML that is the SOAP message. A SOAP message uses a root node of `<soap:Envelope>` that contains the `<soap:Body>` or the body of the SOAP message. Other elements that can be contained in the SOAP message include a SOAP header, `<soap:Header>`, and a SOAP fault — `<soap:Fault>`.

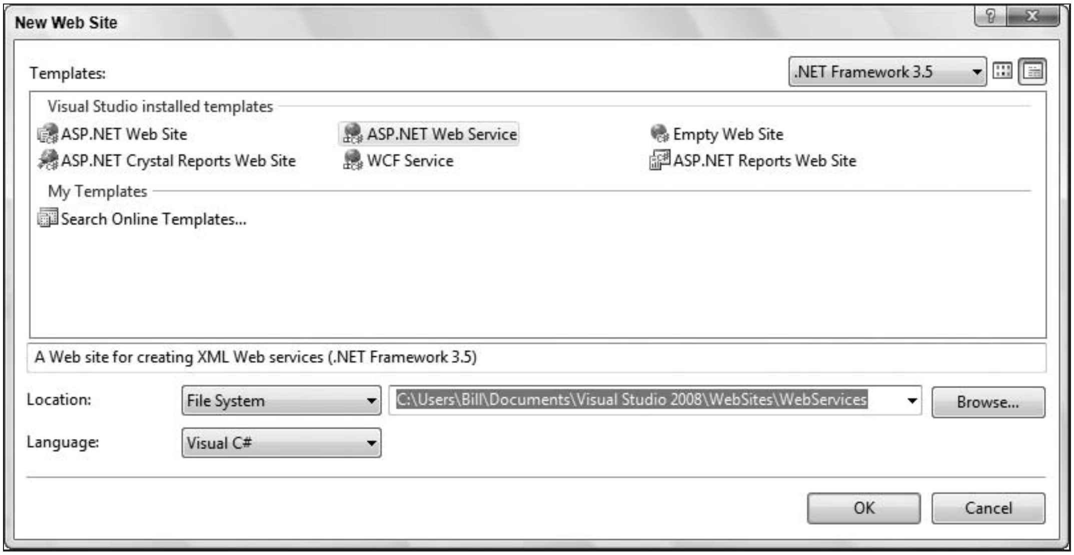
For more information about the structure of a SOAP message, be sure to check out the SOAP specifications. You can find them at the W3C Web site, `www.w3.org/tr/soap`*.*

Building a Simple XML Web Service

Building an XML Web service means that you are interested in exposing some information or logic to another entity either within your organization, to a partner, or to your customers. In a more granular sense, building a Web service means that you, as a developer, simply make one or more methods from a class you create that is enabled for SOAP communication.

You can use Visual Studio 2008 to build an XML Web service. The first step is to actually create a new Web site by selecting File➪New➪Web Site from the IDE menu. The New Web Site dialog opens. Select ASP.NET Web Service, as shown in Figure 29-1.

Visual Studio creates a few files you can use to get started. In the Solution Explorer of Visual Studio (see Figure 29-2) is a single XML Web service named `Service.asmx`; its code-behind file, `Service.vb` or `Service.cs`, is located in the `App_Code` folder.


New Web Site
Templates:
Visual Studio installed templates
ASP.NET Web Site
ASP.NET Web Service
Empty Web Site
ASP.NET Crystal Reports Web Site
WCF Service
ASP.NET Reports Web Site
My Templates
Search Online Templates...
Browse...
OK
Cancel

.NET Framework 3.5

A Web site for creating XML Web services (.NET Framework 3.5)

Location: File System C:\Users\Bill\Documents\Visual Studio 2008\WebSites\WebServices

Language: Visual C#

Figure 29-1

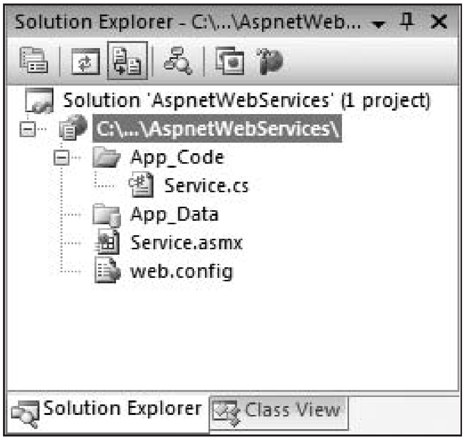

Solution Explorer - C:\...\AspnetWeb...
Solution 'AspnetWebServices' (1 project)
C:\...\AspnetWebServices\
App_Code
Service.cs
App_Data
Service.asmx
web.config
Solution Explorer
Class View

Figure 29-2

Check out the `Service.asmx` file. All ASP.NET Web service files use the `.asmx` file extension instead of the `.aspx` extension used by typical ASP.NET pages.

The WebService Page Directive

Open the `Service.asmx` file in Visual Studio, and you see that the file contains only the `WebService` page directive, as illustrated in Listing 29-3.

Listing 29-3: Contents of the Service.asmx file

```
<%@ WebService Language="VB" CodeBehind="~/App_Code/Service.vb"
    Class="Service" %>
```

You use the `@WebService` directive instead of the `@Page` directive.

The simple `WebService` directive has only four possible attributes. The following list explains these attributes:

- ❑ `Class`: Required. It specifies the class used to define the methods and data types visible to the XML Web service clients.
- ❑ `CodeBehind`: Required only when you are working with an XML Web service file using the code-behind model. It enables you to work with Web services in two separate and more manageable pieces instead of a single file. The `CodeBehind` attribute takes a string value that represents the physical location of the second piece of the Web service — the class file containing all the Web service logic. In ASP.NET, it is best to place the code-behind files in the `App_Code` folder, starting with the default Web service created by Visual Studio when you initially opened the Web service project.
- ❑ `Debug`: Optional. It takes a setting of either `True`or `False`. If the `Debug` attribute is set to `True`, the XML Web service is compiled with debug symbols in place; setting the value to `False` ensures that the Web service is compiled without the debug symbols in place.
- ❑ `Language`: Required. It specifies the language that is used for the Web service.

Looking at the Base Web Service Class File

Now look at the `WebService.vb` or `WebService.cs` file — the code-behind file for the XML Web. By default, a structure of code is already in place in the `WebService.vb` or `WebService.cs` file, as shown in Listing 29-4.

Listing 29-4: Default code structure provided by Visual Studio for your Web service

VB

```
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols

' To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment
' the following line.
' <System.Web.Script.Services.ScriptService()> _
<WebService(Namespace:="http://tempuri.org/")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Public Class Service
     Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function HelloWorld() As String
        Return "Hello World"
    End Function

End Class
```

C#

```
using System;
using System.Linq;
```

Continued

```
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
// To allow this Web Service to be called from script, using ASP.NET AJAX,
// uncomment the following line.
// [System.Web.Script.Services.ScriptService]
public class Service : System.Web.Services.WebService
{
    public Service () {

        //Uncomment the following line if using designed components
        //InitializeComponent();
    }

    [WebMethod]
    public string HelloWorld() {
        return "Hello World";
    }

}
```

Some minor changes to the structure have been made since the .NET 3.5 release. You will notice that the `System.Linq` and `System.Xml.Linq` namespaces are now included in the C# solution. In addition, the other change in this version is the inclusion of the commented `System.Web.Script.Services.ScriptService` object to work with ASP.NET AJAX scripts. To make use of this attribute, you simply uncomment the item.

Since the .NET 1.0/1.1 days, there also have been some big changes. First, the `System.Web.Services.Protocols` namespace is included by default. Therefore, in working with SOAP headers and other capabilities provided via this namespace, you do not need to worry about including it.

The other addition is the new `<WebServiceBinding>` attribute. It builds the XML Web service responses that conform to the WS-I Basic Profile 1.0 release (found at `www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html`).

Besides these minor changes, very little has changed in this basic *Hello World* structure.

Exposing Custom Datasets as SOAP

To build your own Web service example, delete the `Service.asmx` file and create a new file called `Customers.asmx`. This Web service will expose the Customers table from SQL Server. Then jump into the code shown in Listing 29-5.

Listing 29-5: An XML Web service that exposes the Customers table from Northwind

VB

```
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Data
Imports System.Data.SqlClient

<WebService(Namespace := "http://www.wrox.com/customers")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
Public Class Customers
     Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function GetCustomers() As DataSet
        Dim conn As SqlConnection
        Dim myDataAdapter As SqlDataAdapter
        Dim myDataSet As DataSet
        Dim cmdString As String = "Select * From Customers"

        conn = New SqlConnection("Server=localhost;uid=sa;pwd=;database=Northwind")
        myDataAdapter = New SqlDataAdapter(cmdString, conn)

        myDataSet = New DataSet()
        myDataAdapter.Fill(myDataSet, "Customers")

        Return myDataSet
    End Function

End Class
```

C#

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Data;
using System.Data.SqlClient;

[WebService(Namespace = "http://www.wrox.com/customers")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Customers : System.Web.Services.WebService
{

    [WebMethod]
    public DataSet GetCustomers() {
       SqlConnection conn;
       SqlDataAdapter myDataAdapter;
       DataSet myDataSet;
       string cmdString = "Select * From Customers";

       conn = new SqlConnection("Server=localhost;uid=sa;pwd=;database=Northwind");
       myDataAdapter = new SqlDataAdapter(cmdString, conn);

       myDataSet = new DataSet();
       myDataAdapter.Fill(myDataSet, "Customers");

       return myDataSet;
    }

}
```

The WebService Attribute

All Web services are encapsulated within a class. The class is defined as a Web service by the `WebService` attribute placed before the class declaration. Here is an example:

```
<WebService(Namespace := "http://www.wrox.com/customers")> _
```

The `WebService` attribute can take a few properties. By default, the `WebService` attribute is used in your Web service along with the `Namespace` property, which has an initial value of `http://tempuri.org/`. This is meant to be a temporary namespace and should be replaced with a more meaningful and original name, such as the URL where you are hosting the XML Web service. In the example, the `Namespace` value was changed to `www.wrox.com/customers`. Remember that it does not have to be an actual URL; it can be any string value you want. The idea is that it should be unique. It is common practice is to use a URL because a URL is always unique.

Notice that the two languages define their properties within the `WebService` attribute differently. Visual Basic 2008 uses a colon and an equal sign to set the property:

```
Namespace:="http://www.wrox.com/customers"
```

C# uses just an equal sign to assign the properties within the `WebService` attribute values:

```
Namespace="http://www.wrox.com/customers"
```

Other possible `WebService` properties include `Name` and `Description`. `Name` enables you to change how the name of the Web service is presented to the developer via the ASP.NET test page (the test page is discussed a little later in the chapter). `Description` allows you to provide a textual description of the Web service. The description is also presented on the ASP.NET Web service test page. If your `WebService` attribute contains more than a single property, separate the properties using a comma. Here's an example:

```
<WebService(Namespace:="http://www.wrox.com/customers", Name:="GetCustomers")> _
```

The WebMethod Attribute

In Listing 29-5, the class called `Customers` has only a single `WebMethod`. A `WebService` class can contain any number of `WebMethod`s, or a mixture of standard methods along with methods that are enabled to be `WebMethod`s via the use of the attribute preceding the method declaration. The only methods that are accessible across the HTTP wire are the ones to which you have applied the `WebMethod` attribute.

As with the `WebService` attribute, `WebMethod` can also contain some properties, which are described in the following list:

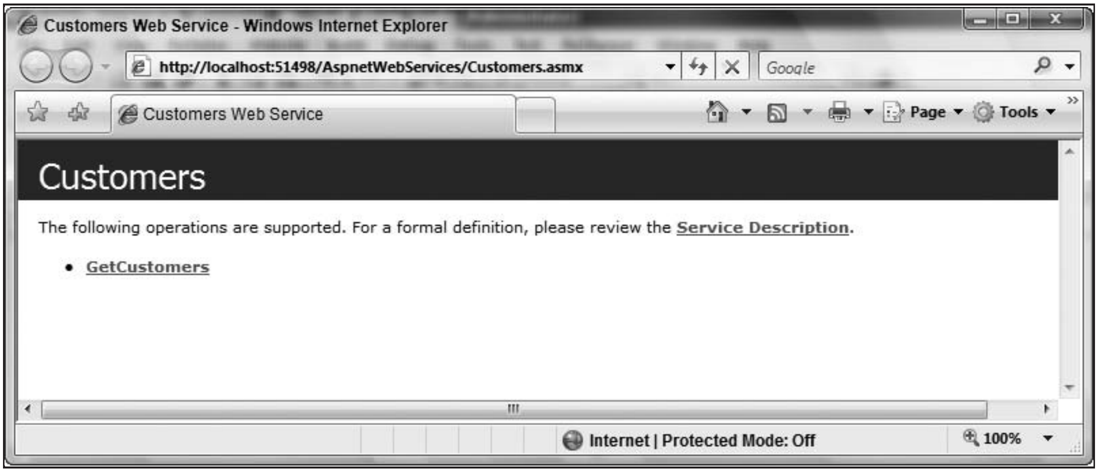
- ❑ `BufferResponse`: When `BufferResponse` is set to `True`, the response from the XML Web service is held in memory and sent as a complete package. If it is set to `False`, the default setting, the response is sent to the client as it is constructed on the server.
- ❑ `CacheDuration`: Specifies the number of seconds that the response should be held in the system's cache. The default setting is `0`, which means that caching is disabled. Putting an XML Web service's response in the cache increases the Web service's performance.

- `Description`: Applies a text description to the `WebMethod` that appears on the `.aspx` test page of the XML Web service.
- `EnableSession`: Setting `EnableSession` to `True` enables session state for a particular `WebMethod`. The default setting is `False`.
- `MessageName`: Applies a unique name to the `WebMethod`. This is a required step if you are working with overloaded `WebMethod`s (discussed later in the chapter).
- `TransactionOption`: Specifies the transactional support for the `WebMethod`. The default setting is `Disabled`. If the `WebMethod` is the root object that initiated the transaction, the Web service can participate in a transaction with another `WebMethod` that requires a transaction. Other possible values include `NotSupported`, `Supported`, `Required`, and `RequiresNew`.

The XML Web Service Interface

The Customers Web service from Listing 29-5 has only a single `WebMethod` that returns a DataSet containing the complete Customers table from the SQL Server Northwind database.

Running `Customers.asmx` in the browser pulls up the ASP.NET Web service test page. This visual interface to your Web service is really meant either for testing purposes or as a reference page for developers interested in consuming the Web services you expose. The page generated for the Customers Web service is shown in Figure 29-3.


Customers Web Service - Windows Internet Explorer
http://localhost:51498/AspnetWebServices/Customers.asmx
Google
Customers Web Service
Page
Tools
Internet | Protected Mode: Off
100%

Customers

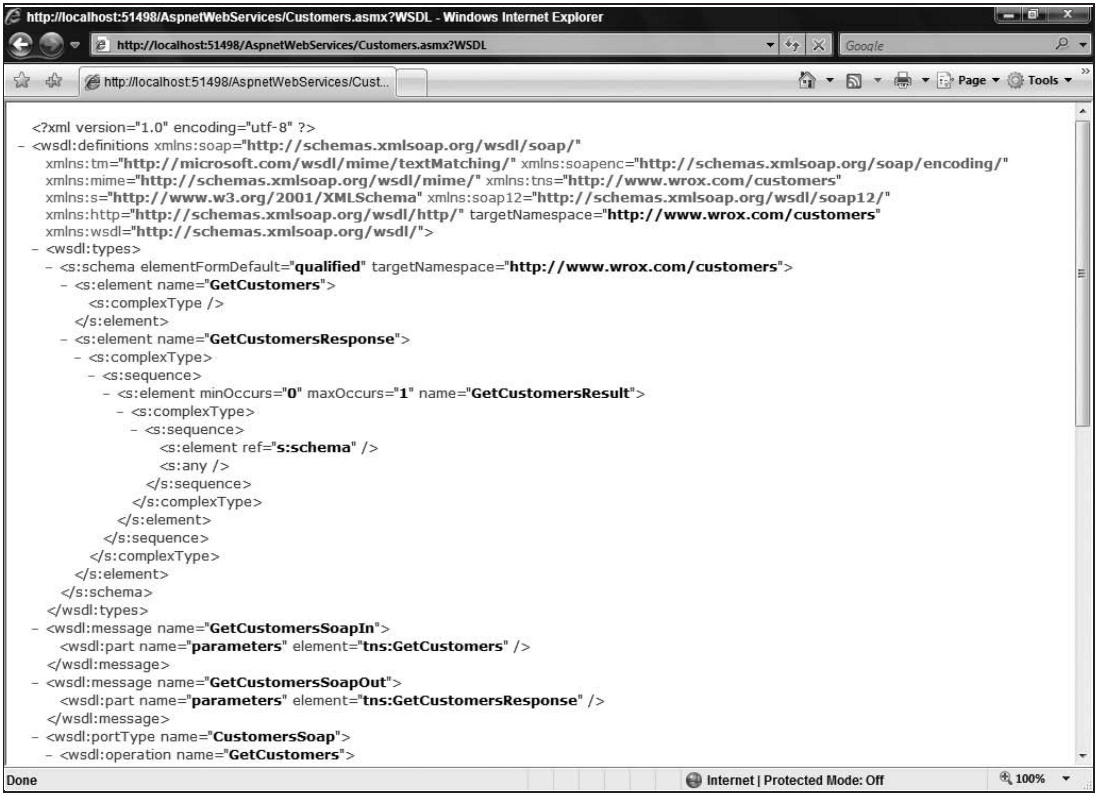
The following operations are supported. For a formal definition, please review the Service Description.

- GetCustomers

Figure 29-3

The interface shows the name of the Web service in the blue bar (the dark bar in this black and white image) at the top of the page. By default, the name of the class is used unless you changed the value through the `Description` property of the `WebService` attribute, as defined earlier. A bulleted list of links to the entire Web service's `WebMethod`s is displayed. In this example, there is only one `WebMethod`: `GetCustomers`.

A link to the Web service's Web Services Description Language (WSDL) document is also available (the link is titled Service Description in the figure). The WSDL file is the actual interface with the Customers Web service. The XML document (shown in Figure 29-4) is not really meant for human consumption; it is designed to work with tools such as Visual Studio, informing the tool what the Web service requires to be consumed. Each Web service requires a request that must have parameters of a specific type. When the request is made, the Web service response comes back with a specific set of data defined using specific data types. Everything you need for the request and a listing of exactly what you are getting back in a response (if you are the consumer) is described in the WSDL document.


http://localhost:51498/AspnetWebServices/Customers.asmx?WSDL - Windows Internet Explorer
http://localhost:51498/AspnetWebServices/Customers.asmx?WSDL
Google
http://localhost:51498/AspnetWebServices/Cust...
Page
Tools
Done
Internet | Protected Mode: Off
100%

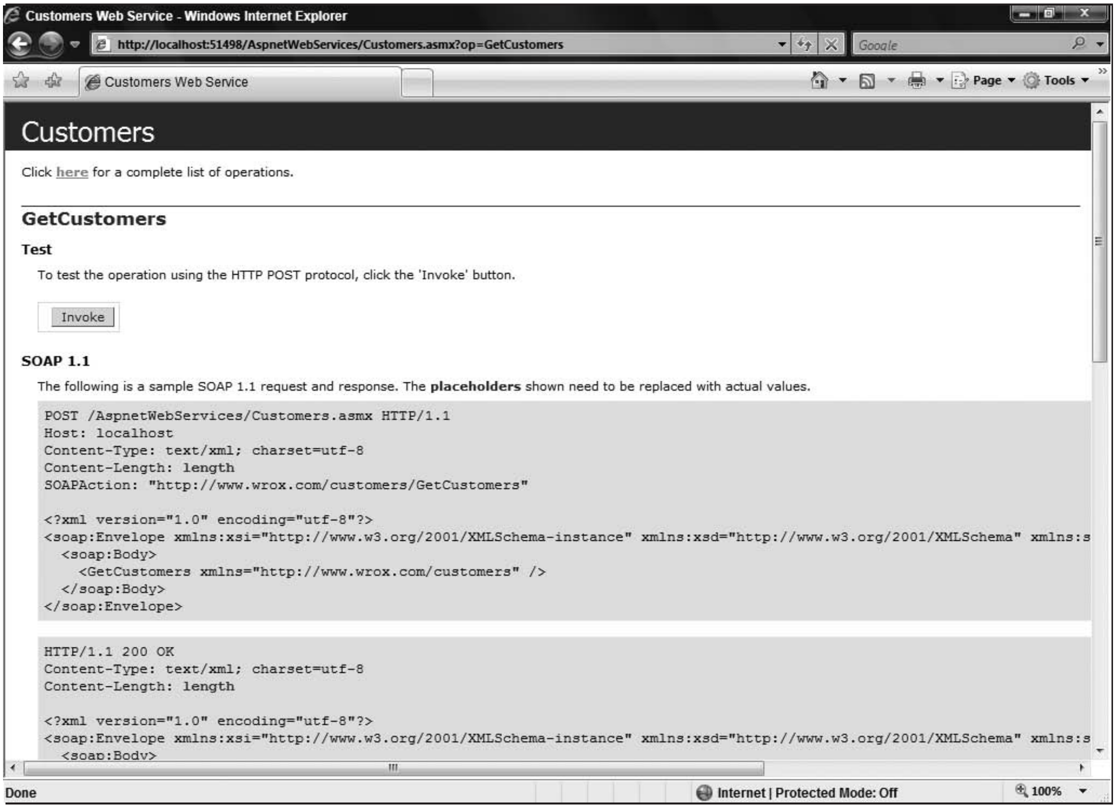
```
  <?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://www.wrox.com/customers"
    xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" targetNamespace="http://www.wrox.com/customers"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  - <wsdl:types>
    - <s:schema elementFormDefault="qualified" targetNamespace="http://www.wrox.com/customers">
      - <s:element name="GetCustomers">
          <s:complexType />
        </s:element>
      - <s:element name="GetCustomersResponse">
        - <s:complexType>
          - <s:sequence>
            - <s:element minOccurs="0" maxOccurs="1" name="GetCustomersResult">
              - <s:complexType>
                - <s:sequence>
                    <s:element ref="s:schema" />
                    <s:any />
                  </s:sequence>
                </s:complexType>
              </s:element>
            </s:sequence>
          </s:complexType>
        </s:element>
      </s:schema>
    </wsdl:types>
  - <wsdl:message name="GetCustomersSoapIn">
      <wsdl:part name="parameters" element="tns:GetCustomers" />
    </wsdl:message>
  - <wsdl:message name="GetCustomersSoapOut">
      <wsdl:part name="parameters" element="tns:GetCustomersResponse" />
    </wsdl:message>
  - <wsdl:portType name="CustomersSoap">
    - <wsdl:operation name="GetCustomers">
```

Figure 29-4

Clicking the `GetCustomers` link gives you a new page, shown in Figure 29-5, that not only describes the `WebMethod` in more detail, but it also allows you to test the `WebMethod` directly in the browser.

At the top of the page is the name of the XML Web service (`Customers`); below that is the name of this particular `WebMethod` (`GetCustomers`). The page shows you the structure of the SOAP messages that are required to consume the `WebMethod`, as well as the structure the SOAP message takes for the response.

Below the SOAP examples is an example of consuming the XML Web service using HTTP Post (with name/value pairs). It is possible to use this method of consumption instead of using SOAP. (This is discussed later in the ‘‘Transport Protocols for Web Services’’ section of this chapter.)


Customers Web Service - Windows Internet Explorer
http://localhost:51498/AspnetWebServices/Customers.asmx?op=GetCustomers
Google
Customers Web Service
Page
Tools
Invoke
Done
Internet | Protected Mode: Off
100%

Customers

Click here for a complete list of operations.

GetCustomers

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /AspnetWebServices/Customers.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.wrox.com/customers/GetCustomers"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:s
  <soap:Body>
    <GetCustomers xmlns="http://www.wrox.com/customers" />
  </soap:Body>
</soap:Envelope>
```

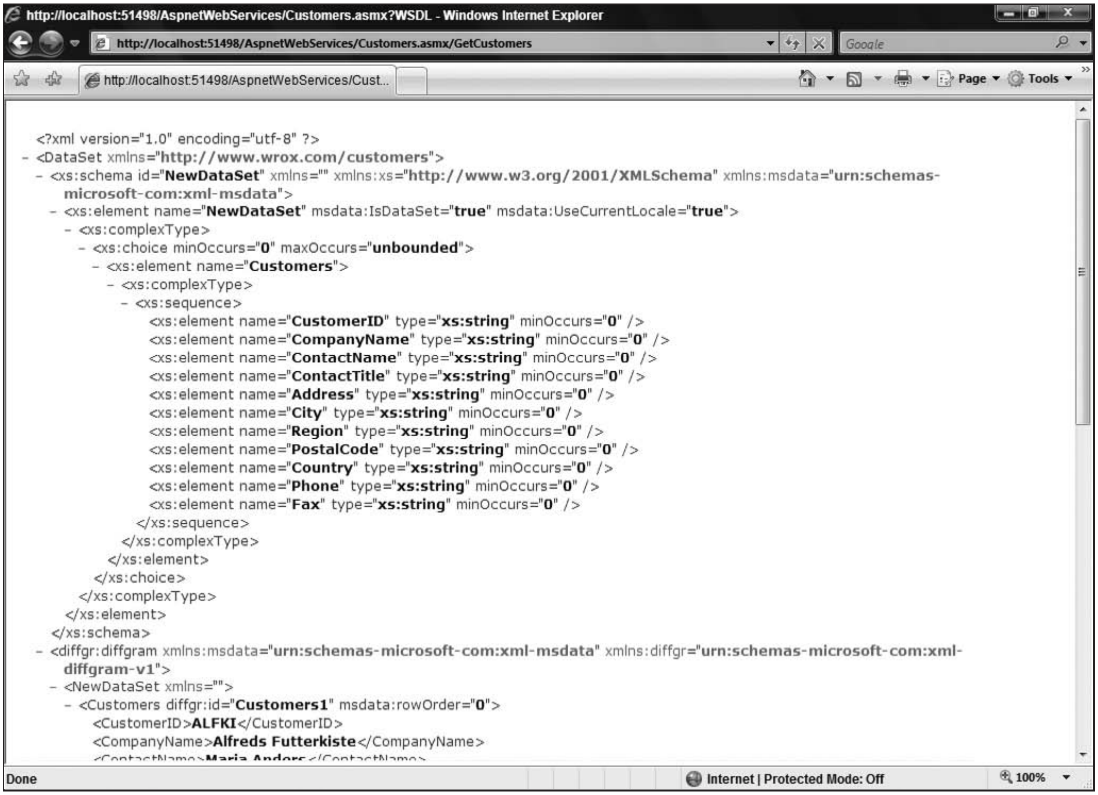
```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:s
  <soap:Body>
```

Figure 29-5

You can test the `WebMethod` directly from the page. In the Test section, you find a form. If the `WebMethod` you are calling requires an input of some parameters to get a response, you see some text boxes included so you can provide the parameters before clicking the Invoke button. If the `WebMethod` you are calling does not require any parameters, you see only the Invoke button and nothing more.

Clicking Invoke is actually sending a SOAP request to the Web service, causing a new browser instance with the result to appear, as illustrated in Figure 29-6.

Now that everything is in place to expose the XML Web service, you can consume it in an ASP.NET application.


http://localhost:51498/AspnetWebServices/Customers.asmx?WSDL - Windows Internet Explorer
http://localhost:51498/AspnetWebServices/Customers.asmx/GetCustomers
Google
http://localhost:51498/AspnetWebServices/Cust...
Page
Tools
Done
Internet | Protected Mode: Off
100%

```
  <?xml version="1.0" encoding="utf-8" ?>
- <DataSet xmlns="http://www.wrox.com/customers">
  - <xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-
      microsoft-com:xml-msdata">
    - <xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
      - <xs:complexType>
        - <xs:choice minOccurs="0" maxOccurs="unbounded">
          - <xs:element name="Customers">
            - <xs:complexType>
              - <xs:sequence>
                  <xs:element name="CustomerID" type="xs:string" minOccurs="0" />
                  <xs:element name="CompanyName" type="xs:string" minOccurs="0" />
                  <xs:element name="ContactName" type="xs:string" minOccurs="0" />
                  <xs:element name="ContactTitle" type="xs:string" minOccurs="0" />
                  <xs:element name="Address" type="xs:string" minOccurs="0" />
                  <xs:element name="City" type="xs:string" minOccurs="0" />
                  <xs:element name="Region" type="xs:string" minOccurs="0" />
                  <xs:element name="PostalCode" type="xs:string" minOccurs="0" />
                  <xs:element name="Country" type="xs:string" minOccurs="0" />
                  <xs:element name="Phone" type="xs:string" minOccurs="0" />
                  <xs:element name="Fax" type="xs:string" minOccurs="0" />
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:choice>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  - <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" xmlns:diffgr="urn:schemas-microsoft-com:xml-
      diffgram-v1">
    - <NewDataSet xmlns="">
      - <Customers diffgr:id="Customers1" msdata:rowOrder="0">
          <CustomerID>ALFKI</CustomerID>
          <CompanyName>Alfreds Futterkiste</CompanyName>
          <ContactName>Maria Anders</ContactName>
```

Figure 29-6

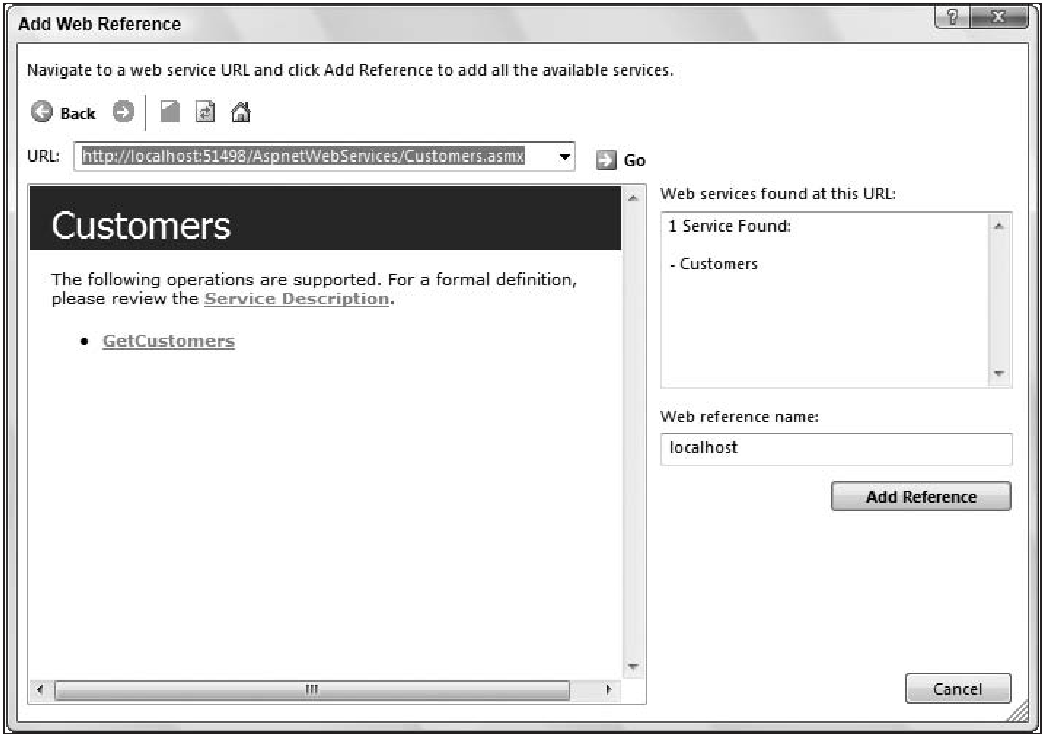
Consuming a Simple XML Web Service

So far, you have seen only half of the XML Web service story. Exposing data and logic as SOAP to disparate systems across the enterprise or across the world is a simple task using .NET and particularly ASP.NET. The other half of the story is the actual consumption of an XML Web service into an ASP.NET application.

You are not limited to consuming XML Web services only into ASP.NET applications; but because this is an ASP.NET book, it focuses on that aspect of the consumption process. Consuming XML Web services into other types of applications is not that difficult and, in fact, is rather similar to how you would consume them using `ASP.NET`. Remember that the Web services you come across can be consumed in Windows Forms, mobile applications, databases, and more. You can even consume XML Web services with other Web services so you can have a single Web service made up of what is basically an aggregate of other Web services.

Adding a Web Reference

To consume the Customers Web service that you created earlier in this chapter, create a new ASP.NET Web site called `CustomerConsumer`. The first step in consuming an XML Web service in an ASP.NET application is to make a reference to the remote object — the Web service. This is done by right-clicking on the root node of your project from within the Solution Explorer of Visual Studio and selecting Add Web Reference. This pulls up the Add Web Reference dialog box, shown in Figure 29-7.



Navigate to a web service URL and click Add Reference to add all the available services.

URL: http://localhost:51498/AspnetWebServices/Customers.asmx

Customers

The following operations are supported. For a formal definition, please review the Service Description.

- GetCustomers

Web services found at this URL:

1 Service Found:

- Customers

Web reference name: localhost

Figure 29-7

The Add Web Reference dialog box enables you to point to a particular `.asmx` file to make a reference to it. Understand that the Add Web Reference dialog box is really looking for WSDL files. Microsoft's XML Web services automatically generate WSDL files based on the `.asmx` files themselves. To pull up the WSDL file in the browser, simply type in the URL of your Web service's `.asmx` file and add a `?WSDL` at the end of the string. For example, you might have the following construction (this is not an actual web service, but simply an example):

```
http://www.wrox.com/MyWebService/Customers.asmx?WSDL
```

Because the Add Web Reference dialog box automatically finds where the WSDL file is for any Microsoft-based XML Web service, you should simply type in the URL of the actual WSDL file for any non–Microsoft-based XML Web service.

If you are using Microsoft's Visual Studio and its built-in Web server instead of IIS, you will be required to also interject the port number the Web server is using into the URL. In this case, your URL would be structured similar to `http://localhost:5444/MyWebService/Customers.asmx?WSDL`*.*

In the Add Web Reference dialog box, change the reference from the default name to something a little more meaningful. If you are working on a single machine, the Web reference might have the name of `localhost`; if you are actually working with a remote Web service, the name is the inverse of the URL, such as `com.wrox.www`. In either case, it is best to rename it so that the name makes a little more sense and is easy to use within your application. In the example here, the Web reference is renamed `WroxCustomers`.

Clicking the Add Reference button causes Visual Studio to make an actual reference to the Web service from the `web.config` file of your application (shown in Figure 29-8). You may find some additional files under the `App_WebReferences` folder — such as a copy of the Web service's WSDL file.

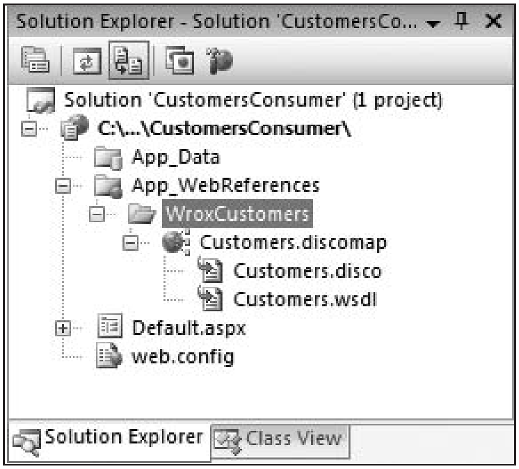

Solution Explorer - Solution 'CustomersCo...
Solution 'CustomersConsumer' (1 project)
C:\...\CustomersConsumer\
App_Data
App_WebReferences
WroxCustomers
Customers.discomap
Customers.disco
Customers.wsdl
Default.aspx
web.config
Solution Explorer
Class View

Figure 29-8

Your consuming application's `web.config` file contains the reference to the Web service in its `<appSettings>` section. The addition is shown in Listing 29-6.

Listing 29-6: Changes to the web.config file after making a reference to the Web service

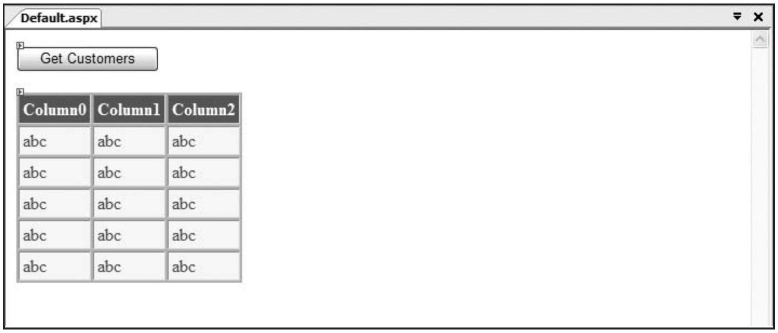
```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
   <appSettings>
      <add key="WroxCustomers.Customers"
       value="http://www.wrox.com/MyWebService/Customers.asmx"/>
   </appSettings>
</configuration>
```

You can see that the `WroxCustomers` reference has been made along with the name of the Web service, providing a key value of `WroxCustomers.Customers`. The `value` attribute takes a value of the location of the Customers Web service, which is found within the `Customers.asmx` page.

Invoking the Web Service from the Client Application

Now that a reference has been made to the XML Web service, you can use it in your ASP.NET application. Create a new Web Form in your project. With this page, you can consume the Customers table from the remote Northwind database directly into your application. The data is placed in a GridView control.

On the design part of the page, place a Button and a GridView control so that your page looks something like the one shown in Figure 29-9.



Column0	Column1	Column2
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc

Figure 29-9

The idea is that, when the end user clicks the button contained on the form, the application sends a SOAP request to the Customers Web service and gets back a SOAP response containing the Customers table, which is then bound to the GridView control on the page. Listing 29-7 shows the code for this simple application.

Listing 29-7: Consuming the Customers Web service in an ASP.NET page

VB

```
<%@ Page Language="VB" %>

<script runat="server">
   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim ws As New WroxCustomers.Customers()
        GridView1.DataSource = ws.GetCustomers()
        GridView1.DataBind()
   End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Web Service Consumer Example</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Button ID="Button1" Runat="server" Text="Get Customers"
         OnClick="Button1_Click" />
        <br />
        <br />
        <asp:GridView ID="GridView1" Runat="server" BorderWidth="1px"
         BackColor="#DEBA84" CellPadding="3" CellSpacing="2" BorderStyle="None"
         BorderColor="#DEBA84">
           <FooterStyle ForeColor="#8C4510" BackColor="#F7DFB5"></FooterStyle>
           <PagerStyle ForeColor="#8C4510" HorizontalAlign="Center"></PagerStyle>
           <HeaderStyle ForeColor="White" Font-Bold="True"
            BackColor="#A55129"></HeaderStyle>
           <SelectedRowStyle ForeColor="White" Font-Bold="True"
            BackColor="#738A9C"></SelectedRowStyle>
```

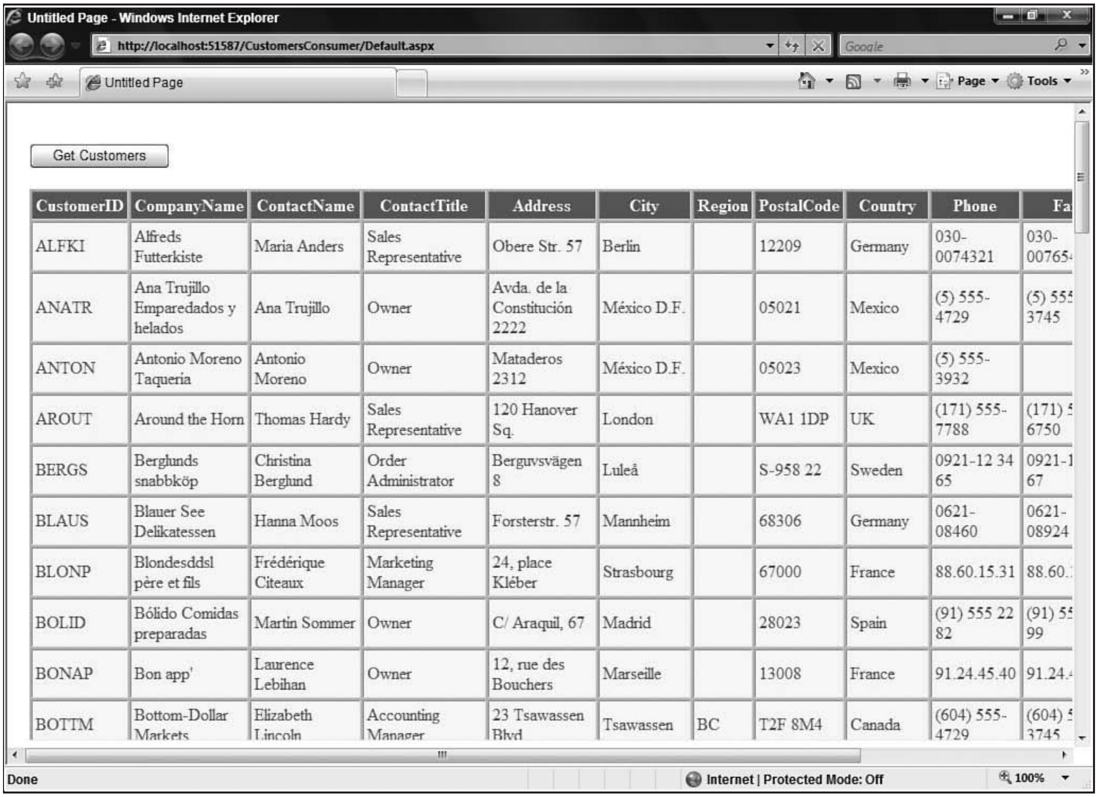
Continued

```
            <RowStyle ForeColor="#8C4510" BackColor="#FFF7E7"></RowStyle>
        </asp:GridView>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(Object sender, EventArgs e) {
       WroxCustomers.Customers ws = new WroxCustomers.Customers();
        GridView1.DataSource = ws.GetCustomers();
        GridView1.DataBind();
    }
</script>
```

The end user is presented with a simple button. Clicking it causes the ASP.NET application to send a SOAP request to the remote XML Web service. The returned DataSet is bound to the GridView control, and the page is redrawn, as shown in Figure 29-10.


Untitled Page - Windows Internet Explorer
http://localhost:51587/CustomersConsumer/Default.aspx
Google
Untitled Page
Page
Tools
Get Customers
Done
Internet | Protected Mode: Off
100%

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fa
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-00765
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555 3745
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK	(171) 555-7788	(171) 6750
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65	0921-1 67
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany	0621-08460	0621-08924
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France	88.60.15.31	88.60.
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67	Madrid		28023	Spain	(91) 555 22 82	(91) 5 99
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille		13008	France	91.24.45.40	91.24.
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd	Tsawassen	BC	T2F 8M4	Canada	(604) 555-4729	(604) 3745

Figure 29-10

The Customers Web service is invoked by the instantiation of the `WroxCustomers.Customers` proxy object:

```
Dim ws As New WroxCustomers.Customers()
```

Then you can use the `ws` object like any other object within your project. In the code example from Listing 29-7, the results of the `ws.GetCustomers()` method call is assigned to the `DataSource` property of the GridView control:

```
GridView1.DataSource = ws.GetCustomers()
```

As you develop or consume more Web services within your applications, you will see more of their power and utility.

Transport Protocols for Web Services

XML Web services use standard wire formats such as HTTP for transmitting SOAP messages back and forth, and this is one of the reasons for the tremendous popularity of Web services. Using HTTP makes using Web services one of the more accessible and consumable messaging protocols when working between disparate systems.

The transport capabilities of Web services are a fresh new addition to the evolutionary idea of a messaging format to use between platforms. DCOM, an older messaging technology that was developed to address the same issues, uses a binary protocol that consists of a method-request layer riding on top of a proprietary communication protocol. One of the problems with using DCOM and similar methods for calling remote objects is that the server's firewall usually gets in the way because DCOM flows through some odd port numbers.

Web services, on the other hand, commonly use a port that is typically open on almost every server — port 80. The port is used for HTTP or Internet traffic. Moving messages from one system to another through port 80 over HTTP is sensible and makes consumption of Web services easy.

An interesting note about XML Web services is that, although many people still think of Web services as SOAP going over HTTP, you can actually consume the Web service in a couple of different ways. Three wire formats are available to Web services: HTTP-GET, HTTP-POST, and SOAP.

Listing 29-8 shows how to work with these different wire formats by consuming a simple Addition Web service.

Listing 29-8: The Addition Web service

VB

```
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols

<WebService(Namespace := "http://www.wrox.com/addition/")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
Public Class WroxMath
```

Continued

```
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function Addition(ByVal a As Integer, ByVal b As Integer) As Integer
        Return (a + b)
    End Function

End Class
```

C#

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://www.wrox.com/addition/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class WroxMath : System.Web.Services.WebService
{

    [WebMethod]
    public int Addition(int a, int b) {
        return a + b;
    }

}
```

The Addition Web service takes two parameters: `a` and `b`. The Web service then adds these numbers and returns the result in a SOAP message. You might typically consume this Web service by sending a request SOAP message to the service. Now look at some of the other means of consumption.

HTTP-GET

The use of HTTP-GET has been rather popular for quite awhile. It enables you to send your entire request, along with any required parameters, all contained within the URL submission. Here is an example of a URL request that is passing a parameter to the server that will respond:

```
http://www.reuters.com?newscategory=world
```

In this example, a request from the `Reuters.com` Web site is made, but in addition to a typical Web request, it is also passing along a parameter. Any parameters that are sent along using HTTP-GET can only be in a name/value pair construction — also known as querystrings. This means that you can have only a single value assigned to a single parameter. You cannot provide hierarchal structures through querystrings. As you can tell from the previous URL construction, the name/value pair is attached to the URL by ending the URL string with a question mark, followed by the variable name.

Using querystrings, you can also pass more than a single name/value pair with the URL request as the following example shows:

```
http://www.reuters.com?newscategory=world&language=en
```

In this example, the URL construction includes two name/value pairs. The name/value pairs are separated with an ampersand (`&`).

Now turn your attention to working with the Addition Web service using HTTP-GET. To accomplish this task, you must enable HTTP-GET from within the Web service application because it is disabled by default.

> *HTTP-GET requests have been disabled by default since ASP.NET 1.1. ASP.NET 1.0 did allow for HTTP-GET and even ran the Web service test interface page using it.*

You should possibly consider using HTTP-GET for non-secure data that you want the end user to get a hold of as simply as possible. To enable HTTP-GET, make changes to your `web.config` file as shown in Listing 29-9.

Listing 29-9: Enabling HTTP-GET in your Web service applications

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
   <system.web>
      <webServices>
         <protocols>
            <add name="HttpGet"/>
         </protocols>
      </webServices>
   </system.web>
</configuration>
```

Creating a `<protocols>` section in your `web.config` file enables you to add or remove protocol communications. For example, you can add missing protocols (such as HTTP-GET) by using the syntax shown previously, or you can remove protocols as the following example shows:

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
   <system.web>
      <webServices>
         <protocols>
            <remove name="HttpGet"/>
            <remove name="HttpPost"/>
            <remove name="HttpSoap"/>
            <remove name="Documentation"/>
         </protocols>
      </webServices>
   </system.web>
</configuration>
```

You do not want to remove everything shown in this code because that would leave your Web service with basically no capability to communicate; but you can see the construction required for any of the protocols that you do want to remove. HTTP-POST and SOAP are covered shortly, but the node removing `Documentation` is interesting in that it can eliminate the ability to invoke the Web services interface test page if you do not want to make that page available.

After you have enabled your Web service to receive HTTP-GET requests, you build a page that uses that protocol to communicate with the Addition Web service. The Web page is shown in Listing 29-10.

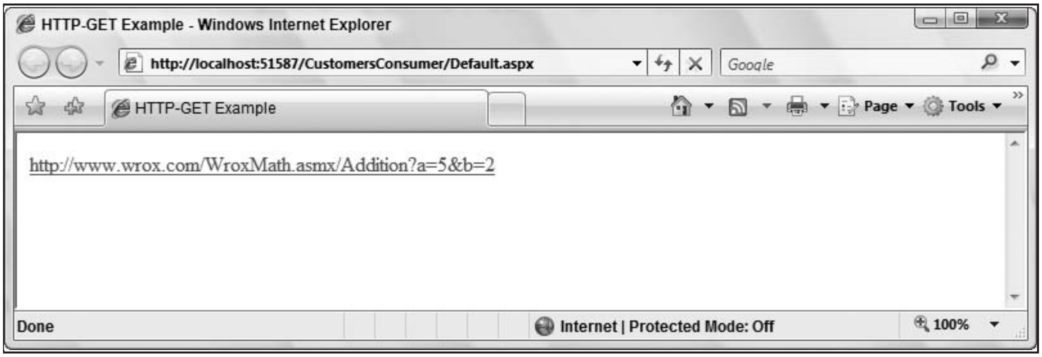
Listing 29-10: Invoking the Addition Web service using HTTP-GET

```
<html>
<head>
   <title>HTTP-GET Example</title>
```

Continued

```
</head>
<body>
    <a href="http://www.wrox.com/WroxMath.asmx/Addition?a=5&b=2">
     http://www.wrox.com/WroxMath.asmx/Addition?a=5&b=2</a>
</body>
</html>
```

This is a simple page with the single hyperlink pointing at the Addition Web service. When the page is run, you get the result shown in Figure 29-11.


HTTP-GET Example - Windows Internet Explorer
http://localhost:51587/CustomersConsumer/Default.aspx
Google
HTTP-GET Example
Page
Tools
Done
Internet | Protected Mode: Off
100%

http://www.wrox.com/WroxMath.asmx/Addition?a=5&b=2

Figure 29-11

To call a Web service using HTTP-GET, you call the actual file (`WroxMath.asmx`), followed by the method name (in this case, `/Addition`), and followed by a querystring list of required parameters. In the example, values for `a` and `b` are passed in the URL. The diagram in Figure 29-12 details the construction of the URL.

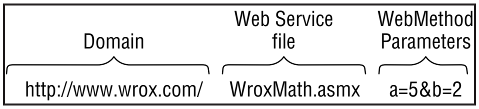

Domain
Web Service file
WebMethod Parameters
http://www.wrox.com/
WroxMath.asmx
a=5&b=2

Figure 29-12

Pull up the `WroxMath.aspx` page and click the link to produce the following text result in the browser:

```
<?xml version="1.0" encoding="utf-8" ?>
<int xmlns="http://www.wrox.com/addition/">7</int>
```

One little caveat when constructing your URL string is that the `WebMethod` name in the URL construction is case-sensitive. If you type `addition` instead of `Addition`, you get an error. Also, be sure to consider when it makes sense to use HTTP-GET; it can be a security risk. It is quite easy to alter values in the querystring to either input false values or values that might cause harm to a server. That's why HTTP-GET capabilities were removed from the default settings of the Web services model.

HTTP-POST

The HTTP-POST protocol is similar to HTTP-GET in that you are sending name/value pairs to the server for processing. The big difference is that HTTP-POST places these name/value pairs within a request header so that they are not visible; HTTP-GET sends these same items within a viewable, open URL string.

Setting up a standard HTML page to communicate with the Addition Web service using HTTP-POST is relatively simple, as illustrated in Listing 29-11.

Listing 29-11: Using HTTP-POST to send a request to an XML Web service

```
<html>
<head>
   <title>HTTP-POST Example</title>
</head>
<body>
   <form method="post" action="http://www.wrox.com/WroxMath.asmx/Addition">
      <p><input type="text" name="a"></p>
      <p><input type="text" name="b"></p>
      <p><input type="submit" value="Call Web Service"></p>
   </form>
</body>
</html>
```

This example puts two text boxes and a button on the form. In order to provide the form elements to be posted in the request, the construction of the text boxes and the `<form>` element are important when working with an XML Web service using HTTP-POST. The `<form>` element here contains two attributes. The first is `method`, which specifies that the form is using HTTP-POST for the request. The `action` attribute provides a link to the `WebMethod` that will be called. As with HTTP-GET, the construction of the URL takes the format of the `.asmx` page followed by the name of the `WebMethod` (`Addition`).

The two text boxes are typical text boxes. This process uses the `name` attribute, giving it a value of the parameter name required by the Web service. In the example, the two required parameters are `a` and `b`.

Posting this page (by clicking the Submit button), produces the same results as the HTTP-GET request:

```
<?xml version="1.0" encoding="utf-8" ?>
<int xmlns="http://tempuri.org/">7</int>
```

Again, this code demonstration is purely an example and you won't find the actual endpoint out there on the Internet.

SOAP

The final method of communicating with an XML Web service is by using SOAP, which was discussed earlier in the chapter. The SOAP message is actually sent in an HTTP request, but does not use the name/value pair construction.

Representing data as SOAP messages brings a lot more value than the simple construction of name/value pairs. SOAP enables you to represent data in a hierarchical manner — something you cannot do when using name/value pairs. For instance, how would you send the Customers table from the Northwind database if you were limited to using name/value pairs? It would be impossible to represent the data properly. SOAP permits this type of data representation. In addition, as you get into more advanced Web service scenarios, you can expand the SOAP messages and allow for authentication/authorization capabilities, SOAP routing, partial encryption capabilities, and more. The expandability of SOAP is a powerful feature.

Web Services Enhancements (WSE) is a powerful toolset from Microsoft that enables you to build advanced Web services for specialized situations, as described previously. You can find more information on the WSE at `http://msdn.microsoft.com/webservices/`*.*

Overloading WebMethods

In the object-oriented world of .NET, it is quite possible to use method overloading in the code you develop. A true object-oriented language has support for *polymorphism* of which method overloading is a part. Method overloading enables you to have multiple methods that use the same name but have different signatures. With method overloading, one method can be called, but the call is routed to the appropriate method based on the full signature of the request. An example of standard method overloading is illustrated in Listing 29-12.

Listing 29-12: Method overloading in .NET

VB

```
Public Function HelloWorld() As String
   Return "Hello"
End Function

Public Function HelloWorld(ByVal FirstName As String) As String
   Return "Hello " & FirstName
End Function
```

C#

```
public string HelloWorld() {
   return "Hello";
}

public string HelloWorld(string FirstName) {
   return "Hello " + FirstName;
}
```

In this example, both methods have the same name, `HelloWorld`. So, which one is called when you invoke `HelloWorld`? Well, it depends on the signature you pass to the method. For instance, you might provide the following:

```
Label1.Text = HelloWorld()
```

This yields a result of just `Hello`. However, you might invoke the `HelloWorld()` method using the following signature:

```
Label1.Text = HelloWorld("Bill Evjen")
```

Then you get back a result of `Hello Bill Evjen`. As you can see, method overloading is a great feature that can be effectively utilized by your ASP.NET applications — but how do you go about overloading `WebMethods`?

If you have already tried to overload any of your WebMethods, you probably got the following error when you pulled up the Web service in the browser:

```
Both System.String HelloWorld(System.String) and System.String HelloWorld() use the
message name 'HelloWorld'. Use the MessageName property of the WebMethod custom
attribute to specify unique message names for the methods.
```

As this error states, the extra step you have to take to overload `WebMethod`s is to use the `MessageName` property. Listing 29-13 shows how.

Listing 29-13: WebMethod overloading in .NET

VB

```
<WebMethod(MessageName:="HelloWorld")> _
Public Function HelloWorld() As String
   Return "Hello"
End Function

<WebMethod(MessageName:="HelloWorldWithFirstName")> _
Public Function HelloWorld(ByVal FirstName As String) As String
   Return "Hello " & FirstName
End Function
```

C#

```
[WebMethod(MessageName="HelloWorld")]
public string HelloWorld() {
   return "Hello";
}

[WebMethod(MessageName="HelloWorldWithFirstName")]
public string HelloWorld(string FirstName) {
   return "Hello " + FirstName;
}
```

In addition to adding the `MessageName` property of the `WebMethod` attribute, you have to disable your Web service's adherence to the WS-I Basic Profile 1.0 specification — which it wouldn't be doing if you perform WebMethod overloading with your Web services. You can disable the conformance to the WS-I Basic Profile specification in a couple of ways. The first way is to add the `<WebServiceBinding>` attribute to your code, as illustrated in Listing 29-14.

Listing 29-14: Changing your Web service so it does not conform to the WS-I Basic Profile spec

VB

```
<WebServiceBinding(ConformsTo := WsiProfiles.None)> _
Public Class MyOverloadingExample
   ' Code here
End Class
```

Continued

C#

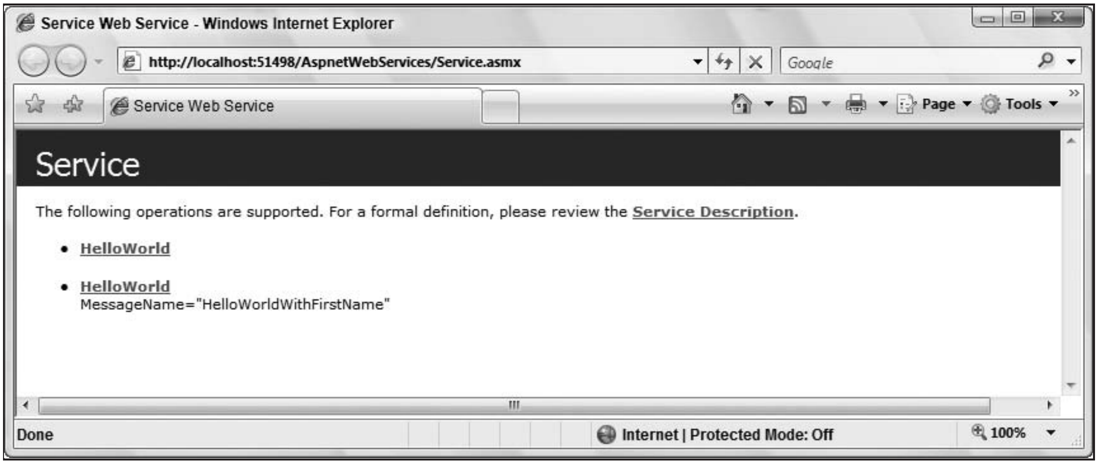
```
[WebServiceBinding(ConformsTo = WsiProfiles.None)]
public class WroxMath : System.Web.Services.WebService
{
   // Code here
}
```

The other option is to turn off the WS-I Basic Profile 1.0 capability in the `web.config` file, as shown in Listing 29-15.

Listing 29-15: Turning off conformance using the web.config file

```
<configuration>
  <system.web>
    <webServices>
      <conformanceWarnings>
        <remove name="BasicProfile1_1" />
      </conformanceWarnings>
    </webServices>
  </system.web>
</configuration>
```

After you have enabled your Web service to overload `WebMethods`, you can see both `WebMethods` defined by their `MessageName` value properties when you pull up the Web service's interface test page in the browser (see Figure 29-13).


Service Web Service - Windows Internet Explorer
http://localhost:51498/AspnetWebServices/Service.asmx
Google
Service Web Service
Page
Tools

Service

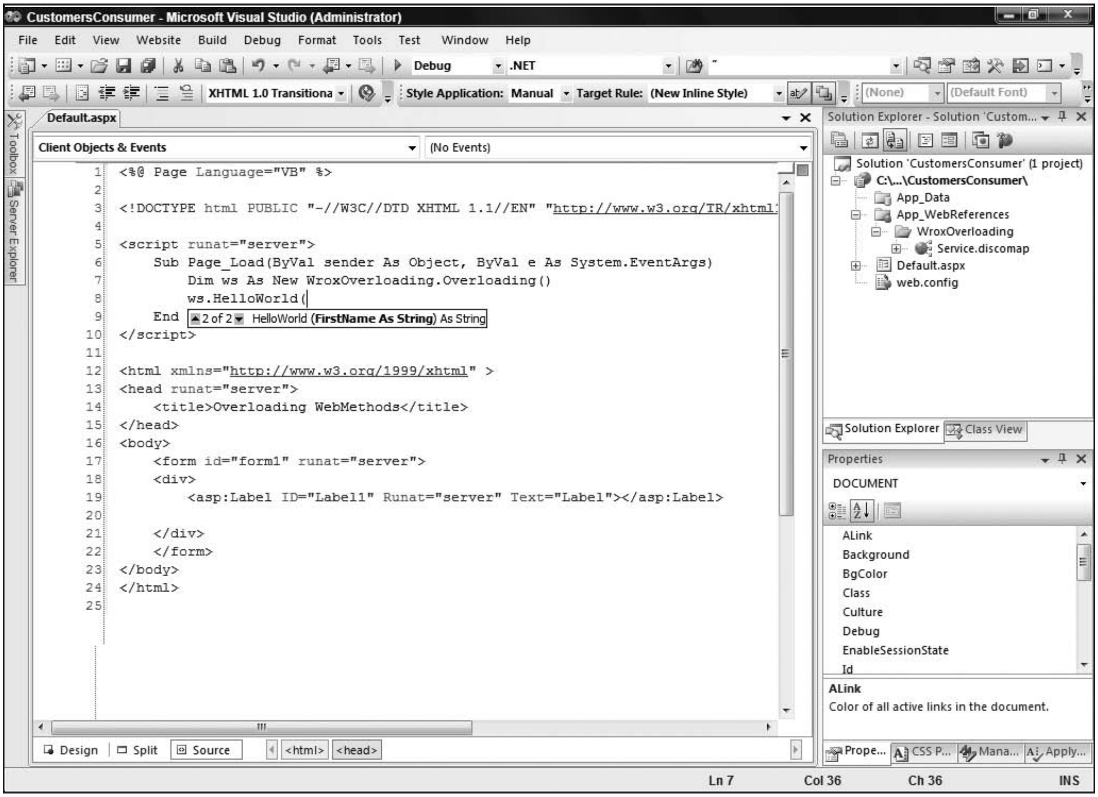
The following operations are supported. For a formal definition, please review the Service Description.

- HelloWorld
- HelloWorld
  MessageName="HelloWorldWithFirstName"

Done
Internet | Protected Mode: Off
100%

Figure 29-13

Although you can see the names of the `WebMethods` as the same, the `MessageName` property shows that they are distinct methods. When the developer consuming the Web service makes a Web reference to your Web service, he will see only a single method name available (in this example, `HelloWorld`). This is shown in the IntelliSense of Visual Studio 2008 in the application consuming these methods (see Figure 29-14).



```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml

<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim ws As New WroxOverloading.Overloading()
        ws.HelloWorld(
    End
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Overloading WebMethods</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" Runat="server" Text="Label"></asp:Label>

    </div>
    </form>
</body>
</html>
```

CustomersConsumer - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Format Tools Test Window Help
Debug
.NET
XHTML 1.0 Transitiona
Style Application: Manual
Target Rule: (New Inline Style)
(None)
(Default Font)
Toolbox
Server Explorer
Default.aspx
Client Objects & Events
(No Events)
2 of 2 HelloWorld (FirstName As String) As String
Design
Split
Source
<html>
<head>
Solution Explorer - Solution 'Custom...
Solution 'CustomersConsumer' (1 project)
C:\...\CustomersConsumer\
App_Data
App_WebReferences
WroxOverloading
Service.discomap
Default.aspx
web.config
Solution Explorer
Class View
Properties
DOCUMENT
ALink
Background
BgColor
Class
Culture
Debug
EnableSessionState
Id
ALink
Color of all active links in the document.
Prope...
CSS P...
Mana...
Apply...
Ln 7
Col 36
Ch 36
INS

Figure 29-14

In the yellow box that pops up to guide developers on the signature structure, you can see two options available — one is an empty signature, and the other requires a single string.

Caching Web Service Responses

Caching is an important feature in almost every application that you build with .NET. Most of the caching capabilities available to you in ASP.NET are discussed in Chapters 19 and 20, but a certain feature of Web services in .NET enables you to cache the SOAP response sent to any of the service's consumers.

First, by way of review, remember that caching is the capability to maintain an in-memory store where data, objects, and various items are stored for reuse. This feature increases the responsiveness of the applications you build and manage. Sometimes, returning cached results can greatly affect performance.

XML Web services use an attribute to control caching of SOAP responses — the `CacheDuration` property. Listing 29-16 shows its use.

Listing 29-16: Utilizing the CacheDuration property

VB

```
<WebMethod(CacheDuration:=60)> _
Public Function GetServerTime() As String
   Return DateTime.Now.ToLongTimeString()
End Function
```

C#

```
[WebMethod(CacheDuration=60)]
public string GetServerTime() {
   return DateTime.Now.ToLongTimeString();
}
```

As you can see, `CacheDuration` is used within the `WebMethod` attribute much like the `Description` and `Name` properties. `CacheDuration` takes an `Integer` value that is equal to the number of seconds during which the SOAP response is cached.

When the first request comes in, the SOAP response is cached by the server, and the consumer gets the same timestamp in the SOAP response for the next minute. After that minute is up, the stored cache is discarded, and a new response is generated and stored in the cache again for servicing all other requests for the next minute.

Among the many benefits of caching your SOAP responses, you will find that the performance of your application is greatly improved when you have a response that is basically re-created again and again without any change.

SOAP Headers

One of the more common forms of extending the capabilities of SOAP messages is to add metadata of the request to the SOAP message itself. The metadata is usually added to a section of the SOAP envelope called the *SOAP header*. Figure 29-15 shows the structure of a SOAP message.

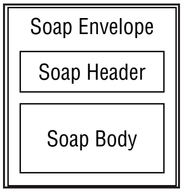


Figure 29-15

The entire SOAP message is referred to as a *SOAP envelope*. Contained within the SOAP message is the *SOAP body* — a piece of the SOAP message that you have been working with in every example thus far. It is a required element of the SOAP message.

The one optional component of the SOAP message is the SOAP header. It is the part of the SOAP message in which you can place any metadata about the overall SOAP request instead of incorporating it in the signature of any of your `WebMethods`. It is important to keep metadata separate from the actual request.

What kind of information? It could include many things. One of the more common items placed in the SOAP header is any authentication/authorization functionality required to consume your Web service or to get at specific pieces of logic or data. Placing usernames and passwords inside the SOAP headers of your messages is a good example of what you might include.

Building a Web Service with SOAP Headers

You can build upon the sample `HelloWorld` Web service that is presented in the default `.asmx` page when it is first pulled up in Visual Studio (from Listing 29-4). Name the new `.asmx` file `HelloSoapHeader.asmx`. The initial step is to add a class that is an object representing what is to be placed in the SOAP header by the client, as shown in Listing 29-17.

Listing 29-17: A class representing the SOAP header

VB

```
Public Class HelloHeader
   Inherits System.Web.Services.Protocols.SoapHeader

   Public Username As String
   Public Password As String
End Class
```

C#

```
public class HelloHeader : System.Web.Services.Protocols.SoapHeader
{
   public string Username;
   public string Password;
}
```

The class, representing a SOAP header object, has to inherit from the `SoapHeader` class from `System.Web.Services.Protocols.SoapHeader`. The `SoapHeader` class serializes the payload of the `<soap:header>` element into XML for you. In the example in Listing 29-17, you can see that this SOAP header requires two elements — simply a username and a password, both of type `String`. The names you create in this class are those used for the subelements of the SOAP header construction, so it is important to name them descriptively.

Listing 29-18 shows the Web service class that instantiates an instance of the `HelloHeader` class.

Listing 29-18: A Web service class that utilizes a SOAP header

VB

```
<WebService(Namespace:="http://www.wrox.com/helloworld")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1, _
  EmitConformanceClaims:=True)> _
Public Class HelloSoapHeader
    Inherits System.Web.Services.WebService
```

Continued

```
    Public myHeader As HelloHeader

    <WebMethod(), SoapHeader("myHeader")> _

    Public Function HelloWorld() As String
        If (myHeader Is Nothing) Then
           Return "Hello World"
        Else
           Return "Hello " & myHeader.Username & ". " & _
              "<br>Your password is: " & myHeader.Password
        End If
    End Function

End Class
```

C#

```
[WebService(Namespace = "http://www.wrox.com/helloworld")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class HelloSoapHeader : System.Web.Services.WebService
{

    public HelloHeader myHeader;

    [WebMethod]
    [SoapHeader("myHeader")]
    public string HelloWorld() {
        if (myHeader == null) {
           return "Hello World";
        }
        else {
           return "Hello " + myHeader.Username + ". " +
              "<br>Your password is: " + myHeader.Password;
        }
    }

}
```

The Web service, `HelloSoapHeader`, has a single `WebMethod` — `HelloWorld`. Within the Web service class, but outside of the `WebMethod` itself, you create an instance of the `SoapHeader` class. This is done with the following line of code:

```
Public myHeader As HelloHeader
```

Now that you have an instance of the `HelloHeader` class that you created earlier called `myHeader`, you can use that instantiation in your `WebMethod`. Because Web services can contain any number of `WebMethod`s, it is not a requirement that all `WebMethod`s use an instantiated SOAP header. You specify whether a `WebMethod` will use a particular instantiation of a SOAP header class by placing the `SoapHeader` attribute before the WebMethod declaration.

```
<WebMethod(), SoapHeader("myHeader")> _
Public Function HelloWorld() As String
   ' Code here
End Function
```

In this example, the `SoapHeader` attribute takes a `string` value of the name of the instantiated `SoapHeader` class — in this case, `myHeader`.

From here, the `WebMethod` actually makes use of the `myHeader` object. If the `myHeader` object is not found (meaning that the client did not send in a SOAP header with his constructed SOAP message), a simple ''`Hello World`'' is returned. However, if values are provided in the SOAP header of the SOAP request, those values are used within the returned `string` value.

Consuming a Web Service Using SOAP Headers

It really is not difficult to build an ASP.NET application that makes a SOAP request to a Web service using SOAP headers. Just as with the Web services that do not include SOAP headers, you make a Web Reference to the remote Web service directly in Visual Studio.

For the ASP.NET page, create a simple page with a single Label control. The output of the Web service is placed in the Label control. The code for the ASP.NET page is shown in Listing 29-19.

Listing 29-19: An ASP.NET page working with an XML Web service using SOAP headers

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim ws As New localhost.HelloSoapHeader()
        Dim wsHeader As New localhost.HelloHeader()

        wsHeader.Username = "Bill Evjen"
        wsHeader.Password = "Bubbles"
        ws.HelloHeaderValue = wsHeader

        Label1.Text = ws.HelloWorld()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Working with SOAP headers</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" Runat="server"></asp:Label>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
```

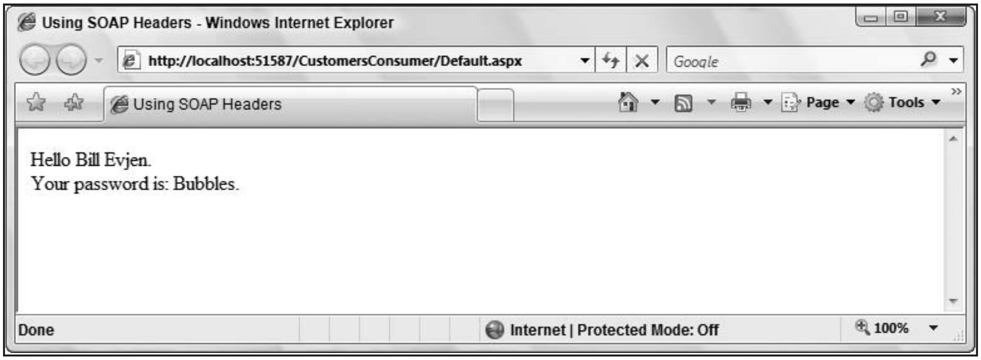
Continued

```
        protected void Page_Load(object sender, System.EventArgs e) {
            localhost.HelloSoapHeader ws = new localhost.HelloSoapHeader();
            localhost.HelloHeader wsHeader = new localhost.HelloHeader();

            wsHeader.Username = "Bill Evjen";
            wsHeader.Password = "Bubbles";
            ws.HelloHeaderValue = wsHeader;

            Label1.Text = ws.HelloWorld();
        }
    </script>
```

Two objects are instantiated. The first is the actual Web service, `HelloSoapHeader`. The second, which is instantiated as `wsHeader`, is the `SoapHeader` object. After both of these objects are instantiated and before making the SOAP request in the application, you construct the SOAP header. This is as easy as assigning values to the `Username` and `Password` properties of the `wsHeader` object. After these properties are assigned, you associate the `wsHeader` object to the `ws` object through the use of the `HelloHeaderValue` property. After you have made the association between the constructed SOAP header object and the actual `WebMethod` object (`ws`), you can make a SOAP request just as you would normally do:

```
Label1.Text = ws.HelloWorld()
```

Running the page produces the result in the browser shown in Figure 29-16.


Using SOAP Headers - Windows Internet Explorer
http://localhost:51587/CustomersConsumer/Default.aspx
Google
Using SOAP Headers
Page
Tools
Done
Internet | Protected Mode: Off
100%

Hello Bill Evjen.
Your password is: Bubbles.

Figure 29-16

What is more interesting, however, is that the SOAP request reveals that the SOAP header was indeed constructed into the overall SOAP message, as shown in Listing 29-20.

Listing 29-20: The SOAP request

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <soap:Header>
      <HelloHeader xmlns="http://www.wrox.com/helloworld/">
         <Username>Bill Evjen</Username>
         <Password>Bubbles</Password>
      </HelloHeader>
   </soap:Header>
   <soap:Body>
      <HelloWorld xmlns="http://www.wrox.com/helloworld/" />
   </soap:Body>
</soap:Envelope>
```

This returns the SOAP response shown in Listing 29-21.

Listing 29-21: The SOAP response

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <soap:Body>
      <HelloWorldResponse xmlns="http://www.wrox.com/helloworld/">
         <HelloWorldResult>Hello Bill Evjen. Your password is:
          Bubbles</HelloWorldResult>
      </HelloWorldResponse>
   </soap:Body>
</soap:Envelope>
```

Requesting Web Services Using SOAP 1.2

Most Web services out there use SOAP version 1.1 for the construction of their messages. With that said, SOAP 1.2 became a W3C recommendation in June 2003 (see `www.w3.org/TR/soap12-part1/`). The nice thing about XML Web services in the .NET Framework platform is that they are capable of communicating in both the 1.1 and 1.2 versions of SOAP.

In an ASP.NET application that is consuming a Web service, you can control whether the SOAP request is constructed as a SOAP 1.1 message or a 1.2 message. Listing 29-22 changes the previous example so that the request uses SOAP 1.2 instead of the default setting of SOAP 1.1.

Listing 29-22: An ASP.NET application making a SOAP request using SOAP 1.2

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim ws As New localhost.HelloSoapHeader()
        Dim wsHeader As New localhost.HelloHeader()

        wsHeader.Username = "Bill Evjen"
        wsHeader.Password = "Bubbles"
        ws.HelloHeaderValue = wsHeader

        ws.SoapVersion = System.Web.Services.Protocols.SoapProtocolVersion.Soap12

        Label1.Text = ws.HelloWorld()
```

Continued

```
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e) {
        localhost.HelloSoapHeader ws = new localhost.HelloSoapHeader();
        localhost.HelloHeader wsHeader = new localhost.HelloHeader();

        wsHeader.Username = "Bill Evjen";
        wsHeader.Password = "Bubbles";
        ws.HelloHeaderValue = wsHeader;

        ws.SoapVersion = System.Web.Services.Protocols.SoapProtocolVersion.Soap12;

        Label1.Text = ws.HelloWorld();
    }
</script>
```

In this example, you first provide an instantiation of the Web service object and use the new `SoapVersion` property. The property takes a value of `System.Web.Services.Protocols.SoapProtocolVersion.Soap12` to work with SOAP 1.2 specifically.

With this bit of code in place, the SOAP request takes the structure shown in Listing 29-23.

Listing 29-23: The SOAP request using SOAP 1.2

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <soap:Header>
      <HelloHeader xmlns="http://www.wrox.com/helloworld/">
         <Username>Bill Evjen</Username>
         <Password>Bubbles</Password>
      </HelloHeader>
   </soap:Header>
   <soap:Body>
      <HelloWorld xmlns="http://www.wrox.com/helloworld/" />
   </soap:Body>
</soap:Envelope>
```

One difference between the two examples is the `xmlns:soap` namespace that is used. The difference actually resides in the HTTP header. When you compare the SOAP 1.1 and 1.2 messages, you see a difference in the `Content-Type` attribute. In addition, the SOAP 1.2 HTTP header does not use the `soapaction` attribute because this is now combined with the `Content-Type` attribute.

You can turn off either SOAP 1.1 or 1.2 capabilities with the Web services that you build by making the proper settings in the `web.config` file, as shown in Listing 29-24.

Listing 29-24: Turning off SOAP 1.1 or 1.2 capabilities

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
   <system.web>
      <webServices>
         <protocols>
            <remove name="HttpSoap"/> <!-- Removes SOAP 1.1 abilities -->
            <remove name="HttpSoap1.2"/> <!-- Removes SOAP 1.2 abilities -->
         </protocols>
      </webServices>
   </system.web>
</configuration>
```

Consuming Web Services Asynchronously

All the Web services that you have been working with in this chapter have been done *synchronously*. This means that after a request is sent from the code of an ASP.NET application, the application comes to a complete standstill until a SOAP response is received.

The process of invoking a `WebMethod` and getting back a result can take some time for certain requests. At times, you are not in control of the Web service from which you are requesting data and, therefore, you are not in control of the performance or response times of these services. For these reasons, you should consider consuming Web services *asynchronously*.

An ASP.NET application that makes an asynchronous request can work on other programming tasks while the initial SOAP request is awaiting a response. When the ASP.NET application is done working on the additional items, it can return to get the result form the Web service.

The great news is that to build an XML Web service that allows asynchronous communication, you don't have to perform any additional actions. All `.asmx` Web services have the built-in capability for asynchronous communication with consumers. The Web service in Listing 29-25 is an example.

Listing 29-25: A slow Web service

VB

```
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols

<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1,
  EmitConformanceClaims:=True)> _
Public Class Async
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function HelloWorld() As String
        System.Threading.Thread.Sleep(1000)
        Return "Hello World"
```

Continued

```
    End Function

End Class
```

C#

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://www.wrox.com/AsyncHelloWorld")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Async : System.Web.Services.WebService
{

    [WebMethod]
    public string HelloWorld() {
        System.Threading.Thread.Sleep(1000);
        return "Hello World";
    }

}
```

This Web service returns a simple `Hello World` as a string, but before it does, the Web service makes a 1000-millisecond pause. This is done by putting the Web service thread to sleep using the `Sleep` method.

Next, take a look at how an ASP.NET application can consume this slow Web service asynchronously, as illustrated in Listing 29-26.

Listing 29-26: An ASP.NET application consuming a Web service asynchronously

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim ws As New localhost.Async()
        Dim myIar As IAsyncResult

        myIar = ws.BeginHelloWorld(Nothing, Nothing)

        Dim x As Integer = 0

        Do Until myIar.IsCompleted = True
            x += 1
        Loop

        Label1.Text = "Result from Web service: " & ws.EndHelloWorld(myIar) & _
           "<br>Local count while waiting: " & x.ToString()
    End Sub
</script>
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Async consumption</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" Runat="server"></asp:Label>
    </div>
    </form>
</body>
</html>
```

C#

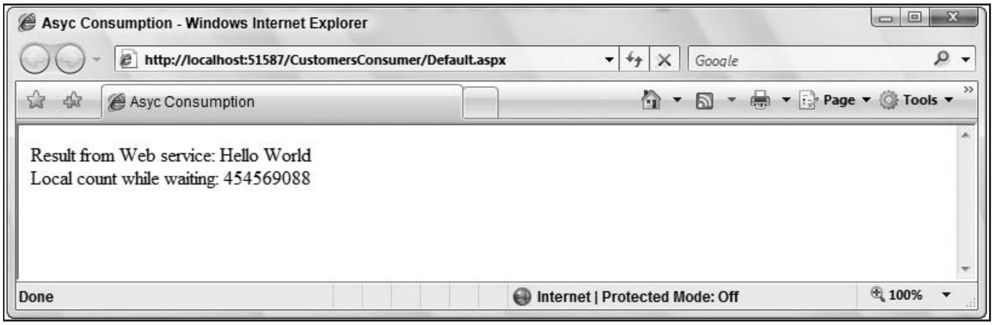
```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e) {
        localhost.Async ws = new localhost.Async();
        IAsyncResult myIar;

        myIar = ws.BeginHelloWorld(null, null);

        int x = 0;

        while (myIar.IsCompleted == false) {
            x += 1;
        }

        Label1.Text = "Result from Web service: " + ws.EndHelloWorld(myIar) +
           "<br>Local count while waiting: " + x.ToString();
    }
</script>
```

When you make the Web reference to the remote Web service in the consuming ASP.NET application, you not only see the `HelloWorld` `WebMethod` available to you in IntelliSense, you also see a `BeginHelloWorld` and an `EndHelloWorld`. To work with the Web service asynchronously, you must utilize the `BeginHelloWorld` and `EndHelloWorld` methods.

Use the `BeginHelloWorld` method to send a SOAP request to the Web service, but instead of the ASP.NET application waiting idly for a response, it moves on to accomplish other tasks. In this case, it is not doing anything that important — just counting the amount of time it is taking in a loop.

After the SOAP request is sent from the ASP.NET application, you can use the `IAsyncResult` object to check whether a SOAP response is waiting. This is done by using `myIar.IsCompleted`. If the asynchronous invocation is not complete, the ASP.NET application increases the value of `x` by one before making the same check again. The ASP.NET application continues to do this until the XML Web service is ready to return a response. The response is retrieved using the `EndHelloWorld` method call.

Results of running this application are similar to what is shown in Figure 29-17.


Asyc Consumption - Windows Internet Explorer
http://localhost:51587/CustomersConsumer/Default.aspx
Google
Asyc Consumption
Page
Tools
Done
Internet | Protected Mode: Off
100%

Result from Web service: Hello World
Local count while waiting: 454569088

Figure 29-17

Windows Communication Foundation

Since the introduction of the .NET Framework 3.0, Microsoft has made available a new way to build Web services beyond the ASP.NET-based Web services presented in this chapter.

Until the .NET Framework 3.0 came out, it was not a simple task to build components that were required to communicate a message from one point to another because Microsoft offered more than one technology that you could use for such an action.

For instance, you could have used ASP.NET Web services (as just seen), Web Service Enhancements 3.0 (WSE), MSMQ, Enterprise Services, .NET Remoting, and even the `System.Messaging` namespace. Each technology has its own pros and cons. ASP.NET Web Services (also known as *ASMX Web Services*) provided the capability to easily build interoperable Web services. The WSE enabled you to easily build services that took advantage of some of the WS-* message protocols. MSMQ enabled the queuing of messages, making it easy to work with solutions that were only intermittently connected. Enterprise Services, provided as a successor to COM+, offered an easy means to build distributed applications. .NET Remoting was a fast way to move messages from one .NET application to another. Moreover, these are Microsoft options only. This does not include all the options available in other environments, such as the Java world.

With so many options available to a Microsoft developer, it can be tough to decide which path to take with the applications you are trying to build. With this in mind, Microsoft has created the Windows Communication Foundation (WCF).

WCF is a new framework for building service-oriented applications. Microsoft wanted to provide its developers with a framework to quickly get a proper service-oriented architecture up and running. Using the WCF, you will be able to take advantage of all the items that make distribution technologies powerful. WCF is the answer and the successor to all these other message distribution technologies.

The Larger Move to SOA

Upon examining WCF, you will find that it is part of a larger movement that organizations are making toward the much talked about *service-oriented architecture*, or *SOA*. An SOA is a message-based service architecture that is vendor agnostic. As a result, you have the capability to distribute messages across a system, and the messages are interoperable with other systems that would otherwise be considered incompatible with the provider system.

Looking back, you can see the gradual progression to the service-oriented architecture model. In the 1980s, the revolution arrived with the concept of everything being an object. When object-oriented programming came on the scene, it was enthusiastically accepted as the proper means to represent entities within a programming model. The 1990s took that one step further, and the component-oriented model was born. This enabled objects to be encapsulated in a tightly coupled manner. It was only recently that the industry turned to a service-oriented architecture because developers and architects needed to take components and have them distributed to other points in an organization, to their partners, or to their customers. This distribution system needed to have the means to transfer messages between machines that were generally incompatible with one another. In addition, the messages had to include the ability to express the metadata about how a system should handle a message.

If you ask 10 people what an SOA is, you'll probably get 11 different answers, but there are some common principles that are considered to be foundations of a service-oriented architecture:

- ☐ **Boundaries are explicit:** Any datastore, logic, or entity uses an interface to expose its data or capabilities. The interface provides the means to hide the behaviors within the service, and the interface front-end enables you to change this behavior as required without affecting downstream consumers.
- ☐ **Services are autonomous:** All the services are updated or versioned independently of one another. Thus, you do not upgrade a system in its entirety; instead, each component of these systems is an individual entity within itself and can move forward without waiting for other components to progress forward. Note that with this type of model, once you publish an interface, that interface must remain unchanged. Interface changes require new interfaces (versioned, of course).
- ☐ **Services are based upon contracts, schemas, and policies:** All services developed require a contract regarding what is required to consume items from the interface (usually done through a WSDL document). Along with a contract, schemas are required to define the items passed in as parameters or delivered through the service (using XSD schemas). Finally, policies define any capabilities or requirements of the service.
- ☐ **Service compatibility that is based upon policy:** The final principle enables services to define policies (decided at runtime) that are required to consume the service. These policies are usually expressed through WS-Policy.

If your own organization is considering establishing an SOA, the WCF is a framework that works on these principles and makes it relatively simple to implement. The next section looks at what the WCF offers. Then you can dive into building your first WCF service.

WCF Overview

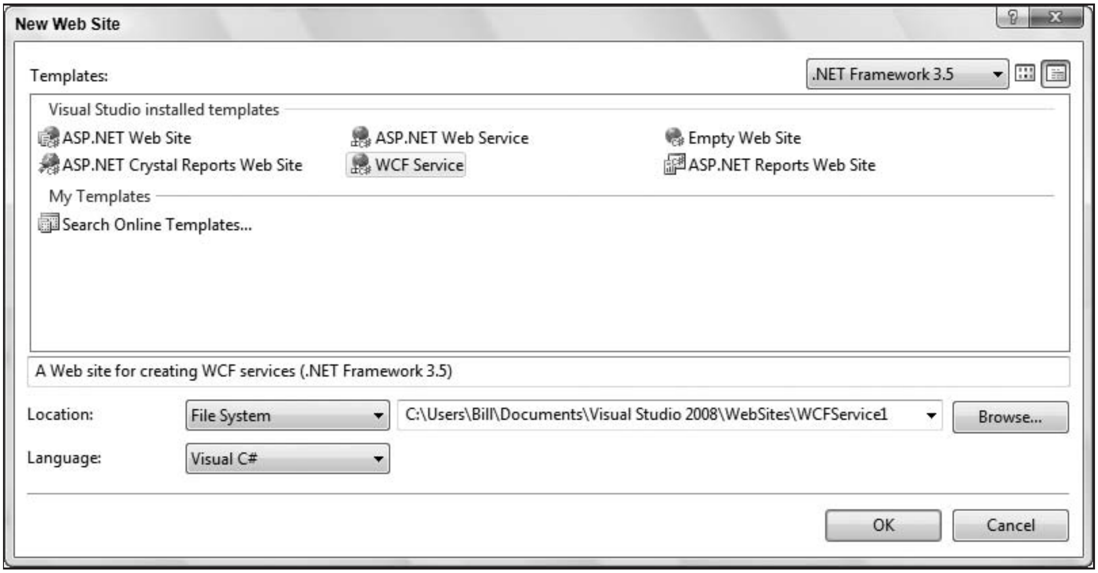
As stated, the Windows Communication Foundation is a means to build distributed applications in a Microsoft environment. Although the distributed application is built upon that environment, this does not mean that consumers are required to be Microsoft clients or to take any Microsoft component or technology to accomplish the task of consumption. On the other hand, building WCF services means you are also building services that abide by the principles set forth in the aforementioned SOA discussion and that these services are vendor agnostic — thus, they are able to be consumed by almost anyone.

You can build WCF services using Visual Studio 2008. Note that because this is a .NET Framework 3.0 component, you are actually limited to the operating systems in which you can run a WCF service. Whereas the other Microsoft distribution technologies mentioned in this chapter do not have too many limitations on running on Microsoft operating systems, an application built with WCF can only run on Windows XP SP2, Windows Vista, or Windows Server 2008.

Building a WCF Service

Building a WCF service is not hard to accomplish. If you are working from a .NET Framework 2.0 environment, you need to install the .NET Framework 3.0. If you have installed the . NET Framework 3.5, then you will find that both the .NET Framework 2.0 and 3.0 have been installed also.

From there, it is easy to build WCF services directly in Visual Studio 2008 because it is already geared to work with this application type. If you are working with Visual Studio 2005, you need to install the Visual Studio 2005 extensions for .NET Framework 3.0 (WCF and WPF). Download these Visual Studio extensions, if you are using Visual Studio 2005. Installing the extensions into Visual Studio 2005 adds a WCF project to your IDE. If you are using Visual Studio 2008, the view of the project from the New Web Site dialog box is presented in Figure 29-18.



.NET Framework 3.5

A Web site for creating WCF services (.NET Framework 3.5)

Location: File System C:\Users\Bill\Documents\Visual Studio 2008\WebSites\WCFService1

Language: Visual C#

Figure 29-18

When you build a WCF project in this manner, the idea is that you build a traditional class library that is compiled down to a DLL that can then be added to another project. The separation of code and project is a powerful division on larger projects. That said, you can, however, just as easily build a WCF service directly in your .NET project, whether that is a console application or a Windows Forms application. The approach taken for the examples in this chapter show you how to build a WCF service that is hosted in a console application. Keep in mind that for the services you actually build and deploy, it is usually better to build them directly as a WCF Service Library project and use the created DLL in your projects or in IIS itself.

Before we jump into building a WCF service, first consider what makes up a service built upon the WCF framework.

What Makes a WCF Service

When looking at a WCF service, it is important to understand that it is made up of three parts: the service, one or more endpoints, and an environment in which to host the service.

A service is a class that is written in one of the .NET-compliant languages. The class can contain one or more methods that are exposed through the WCF service. A service can have one or more endpoints. An endpoint is used to communicate through the service to the client.

Endpoints themselves are also made up of three parts. These parts are usually defined by Microsoft as the ABC of WCF. Each letter of WCF means something in particular in the WCF model, including the following:

- ☐ ''A'' is for address
- ☐ ''B'' is for binding
- ☐ ''C'' is for contract

Basically, you can think of this as follows: ''A'' is the *where*, ''B'' is the *how*, and ''C'' is the *what*. Finally, a hosting environment is where the service is contained. This constitutes an application domain and process. All three of these elements (the service, the endpoints, and the hosting environment) are put together to create a WCF service offering, as depicted in Figure 29-19.

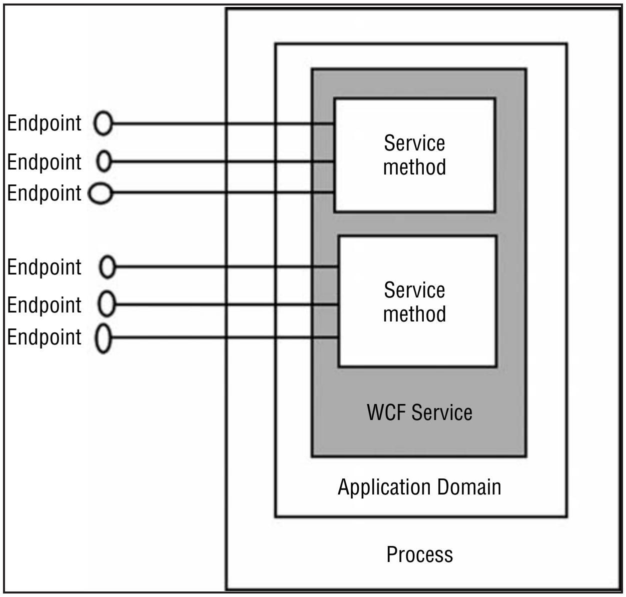


Figure 29-19

The next step is to create a basic service using the WCF framework.

Creating Your First WCF Service

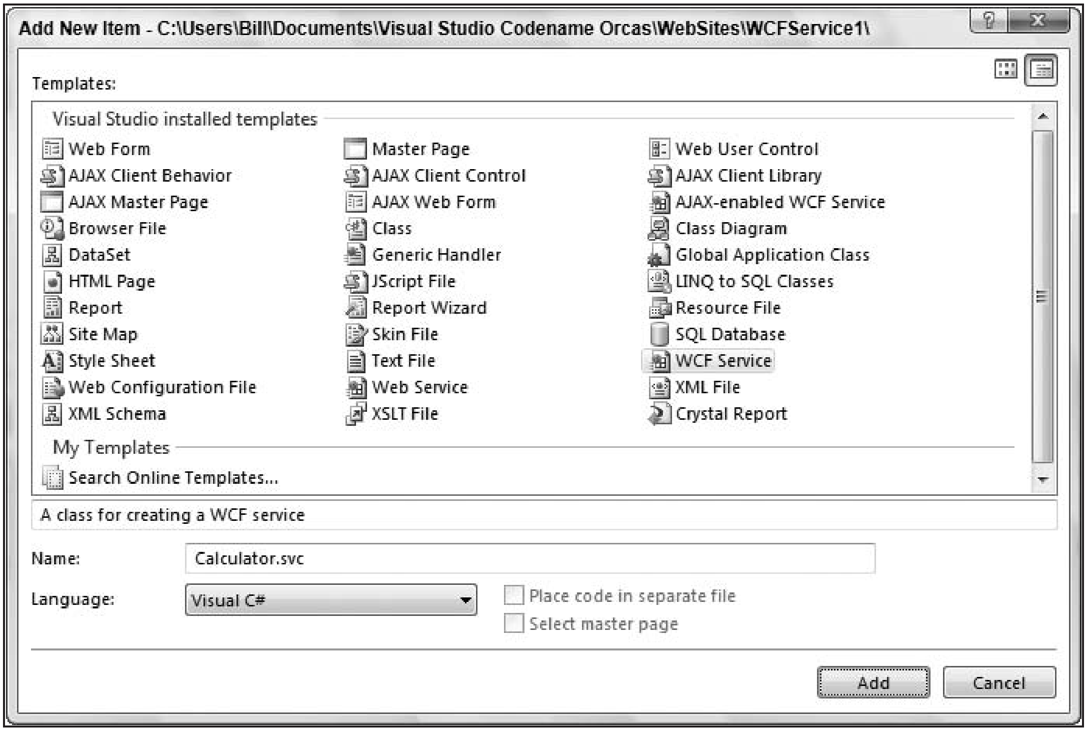
To build your service, prior to hosting it, you must perform two main steps. First, you need to create a service contract. Second, you must create a data contract. The service contract is really a class with the methods that you want to expose from the WCF service. The data contract is a class that specifies the structure you want to expose from the interface.

Once you have a service class in place, you can host it almost anywhere. When running this from Visual Studio 2008, you will be able to use the same built-in hosting mechanisms that are used by any standard ASP.NET application. To build your first WCF application, select File➪New➪Web Site from the Visual Studio 2008 menu and call the project `WCFService1`.

The example this chapter will run through here demonstrates how to build the WCF service by building the interface, followed by the service itself.

Creating the service framework

The first step is to create the services framework in the project. To do this, right-click on the project and select Add New Item from the provided menu. From this dialog box, select WCF Service, and name the service Calculator.svc, as illustrated in Figure 29-20.


Add New Item - C:\Users\Bill\Documents\Visual Studio Codename Orcas\WebSites\WCFService1\
Templates:
Visual Studio installed templates
Web Form
AJAX Client Behavior
AJAX Master Page
Browser File
DataSet
HTML Page
Report
Site Map
Style Sheet
Web Configuration File
XML Schema
Master Page
AJAX Client Control
AJAX Web Form
Class
Generic Handler
JScript File
Report Wizard
Skin File
Text File
Web Service
XSLT File
Web User Control
AJAX Client Library
AJAX-enabled WCF Service
Class Diagram
Global Application Class
LINQ to SQL Classes
Resource File
SQL Database
WCF Service
XML File
Crystal Report
My Templates
Search Online Templates...
Add
Cancel

A class for creating a WCF service

Name: Calculator.svc

Language: Visual C#

☐ Place code in separate file

☐ Select master page

Figure 29-20

This step creates a `Calculator.svc` file, a `Calculator.cs` file, and an `ICalculator.cs` file. The `Calculator.svc` file is a simple file that includes only the page directive, whereas the `Calculator.cs` does all the heavy lifting. The `Calculator.cs` file is an implementation of the `ICalculator.cs` interface.

Working with the Interface

To create your service you need a service contract. The service contract is the interface of the service. This consists of all the methods exposed, as well as the input and output parameters that are required to invoke the methods. To accomplish this task, turn to the `ICalculator.vb` or `ICalculator.cs` (depending on the language you are using). The interface you need to create is presented in Listing 29-27.

Listing 29-27: Creating the interface

VB

```
Imports System
Imports System.ServiceModel

<ServiceContract()> _
Public Interface ICalculator
    <OperationContract()> _
    Function Add(ByVal a As Integer, ByVal b As Integer) As Integer

    <OperationContract()> _
    Function Subtract(ByVal a As Integer, ByVal b As Integer) As Integer

    <OperationContract()> _
    Function Multiply(ByVal a As Integer, ByVal b As Integer) As Integer

    <OperationContract()> _
    Function Divide(ByVal a As Integer, ByVal b As Integer) As Integer
End Interface
```

C#

```
using System.ServiceModel;

[ServiceContract]
public interface ICalculator
{
    [OperationContract]
    int Add(int a, int b);

    [OperationContract]
    int Subtract(int a, int b);

    [OperationContract]
    int Multiply(int a, int b);

    [OperationContract]
    int Divide(int a, int b);
}
```

This is pretty much the normal interface definition you would expect, but with a couple of new attributes included. To gain access to these required attributes, you need to make a reference to the `System.ServiceModel` namespace. This will give you access to the `<ServiceContract()>` and `<OperationContract()>` attributes.

The `<ServiceContract()>` attribute is used to define the class or interface as the service class, and it needs to precede the opening declaration of the class or interface. In this case, the example in the preceding code is based upon an interface:

```
<ServiceContract()> _
Public Interface ICalculator

   ' Code removed for clarity

End Interface
```

Within the interface, four methods are defined. Each method is going to be exposed through the WCF service as part of the service contract. For this reason, each method is required to have the `<OperationContract()>` attribute applied.

```
<OperationContract()> _
Function Add(ByVal a As Integer, ByVal b As Integer) As Integer
```

Utilizing the Interface

The next step is to create a class that implements the interface. Not only is the new class implementing the defined interface, it is also implementing the service contract. For this example, add this class to the same `Calculator.vb` or `.cs` file. The following code, illustrated in Listing 29-28, shows the implementation of this interface.

Listing 29-28: Implementing the interface

VB

```
Public Class Calculator
    Implements ICalculator

    Public Function Add(ByVal a As Integer, ByVal b As Integer) As Integer _
       Implements ICalculator.Add

        Return (a + b)
    End Function

    Public Function Subtract(ByVal a As Integer, ByVal b As Integer) As Integer _
       Implements ICalculator.Subtract

        Return (a - b)
    End Function

    Public Function Multiply(ByVal a As Integer, ByVal b As Integer) As Integer _
       Implements ICalculator.Multiply

        Return (a * b)
    End Function

    Public Function Divide(ByVal a As Integer, ByVal b As Integer) As Integer _
```

```
        Implements ICalculator.Divide

         Return (a / b)
     End Function
End Class
```

C#

```
public class Calculator : ICalculator
{
   public int Add(int a, int b)
   {
       return (a + b);
   }

    public int Subtract(int a, int b)
    {
        return (a - b);
    }

    public int Multiply(int a, int b)
    {
        return (a * b);
    }

    public int Divide(int a, int b)
    {
        return (a / b);
    }
}
```

From these new additions, you can see that you don't have to do anything different to the `Calculator` class. It is a simple class that implements the `ICalculator` interface and provides implementations of the `Add()`, `Subtract()`, `Multiply()`, and `Divide()` methods.

With the interface and the class available, you now have your WCF service built and ready to go. The next step is to get the service hosted. Note that this is a simple service — it exposes only simple types, rather than a complex type. This enables you to build only a service contract and not have to deal with the construction of a data contract. The construction of data contracts is presented later in this chapter.

Hosting the WCF Service in a Console Application

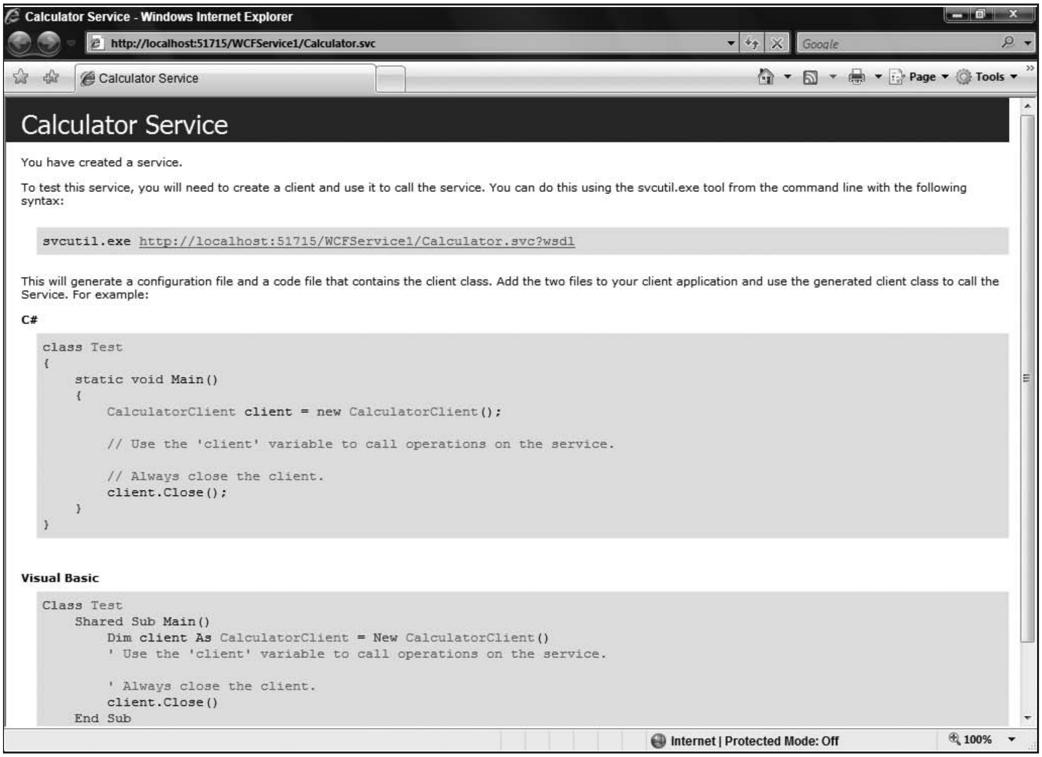
The next step is to take the service just developed and host it in some type of application process. You have many available hosting options, including the following:

- ❑ Console applications
- ❑ Windows Forms applications
- ❑ Windows Presentation Foundation applications

- Managed Windows Services
- Internet Information Services (IIS) 5.1
- Internet Information Services (IIS) 6.0
- Internet Information Services (IIS) 7.0 and the Windows Activation Service (WAS)

As stated earlier, this example hosts the service in the developer Web server provided by Visual Studio 2008. There are a couple of ways to activate hosting — either through the direct coding of the hosting behaviors or through declarative programming (usually done via the configuration file).

Compiling and running this application produces the results illustrated in Figure 29-21.


Calculator Service - Windows Internet Explorer
http://localhost:51715/WCFService1/Calculator.svc
Google
Calculator Service
Page
Tools
Internet | Protected Mode: Off
100%

Calculator Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil.exe tool from the command line with the following syntax:

```
svcutil.exe http://localhost:51715/WCFService1/Calculator.svc?wsdl
```

This will generate a configuration file and a code file that contains the client class. Add the two files to your client application and use the generated client class to call the Service. For example:

C#

```
class Test
{
    static void Main()
    {
        CalculatorClient client = new CalculatorClient();

        // Use the 'client' variable to call operations on the service.

        // Always close the client.
        client.Close();
    }
}
```

Visual Basic

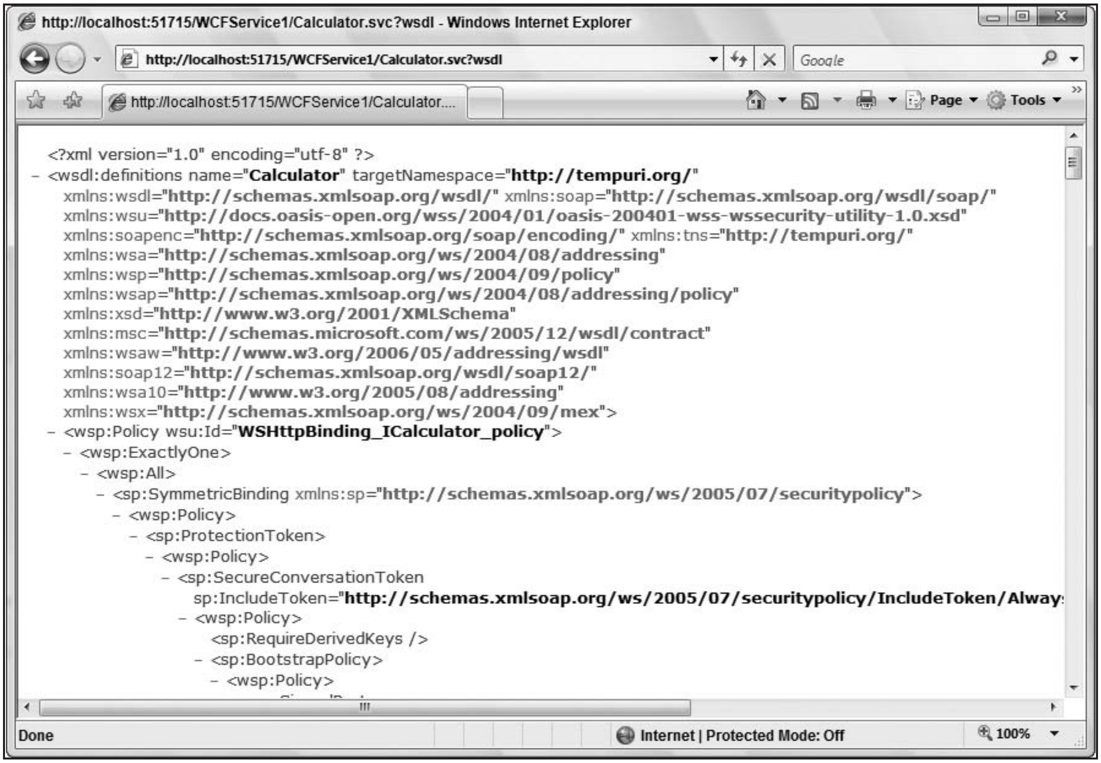
```
Class Test
    Shared Sub Main()
        Dim client As CalculatorClient = New CalculatorClient()
        ' Use the 'client' variable to call operations on the service.

        ' Always close the client.
        client.Close()
    End Sub
```

Figure 29-21

You will notice that this is quite similar to how it appears when you build an ASP.NET Web service.

Reviewing the WSDL Document

The page presented in Figure 29-21 was the information page about the service. In the image, notice that there is also a link to the WSDL file of the service. As with ASP.NET Web services, you find that a WCF service can also auto-generate the WSDL file. Clicking on the WSDL link shows the WSDL in the browser, as illustrated in Figure 29-22.


http://localhost:51715/WCFService1/Calculator.svc?wsdl - Windows Internet Explorer
http://localhost:51715/WCFService1/Calculator.svc?wsdl
Google
http://localhost:51715/WCFService1/Calculator....
Page
Tools
Done
Internet | Protected Mode: Off
100%

```
  <?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions name="Calculator" targetNamespace="http://tempuri.org/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns="http://tempuri.org/"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:wsap="http://schemas.xmlsoap.org/ws/2004/08/addressing/policy"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:msc="http://schemas.microsoft.com/ws/2005/12/wsdl/contract"
    xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:wsa10="http://www.w3.org/2005/08/addressing"
    xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex">
  - <wsp:Policy wsu:Id="WSHttpBinding_ICalculator_policy">
    - <wsp:ExactlyOne>
      - <wsp:All>
        - <sp:SymmetricBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          - <wsp:Policy>
            - <sp:ProtectionToken>
              - <wsp:Policy>
                - <sp:SecureConversationToken
                    sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/Alway
                  - <wsp:Policy>
                      <sp:RequireDerivedKeys />
                    - <sp:BootstrapPolicy>
                      - <wsp:Policy>
```

Figure 29-22

With this WSDL file, you can consume the service it defines through an HTTP binding. Note the following element at the bottom of the document, as shown in Listing 29-29.

Listing 29-29: The part of the WSDL file showing the service's endpoint

```
<wsdl:service name="Calculator">
   <wsdl:port name="WSHttpBinding_ICalculator"
    binding="tns:WSHttpBinding_ICalculator">
      <soap12:address
       location="http://localhost:51715/WCFService1/Calculator.svc" />
      <wsa10:EndpointReference>
<wsa10:Address>http://localhost:51715/WCFService1/Calculator.svc
         </wsa10:Address>
         <Identity
xmlns="http://schemas.xmlsoap.org/ws/2006/02/addressingidentity">
         <Upn>Lipper-STL-LAP\Bill</Upn>
         </Identity>
      </wsa10:EndpointReference>
   </wsdl:port>
</wsdl:service>
```

This element in the XML document indicates that in order to consume the service, the end user needs to use SOAP 1.2 over HTTP. This is presented through the use of the `<soap12:address>` element in the document. The `<wsa10:EndpointReference>` is a WS-Addressing endpoint definition.

Using this simple WSDL document, you can now build a consumer that makes use of this interface.

Building the WCF Consumer

Now that an HTTP service is out there, which you built using the WCF framework, the next step is to build a consumer application in ASP.NET that uses the simple `Calculator` service. The consumer sends its request via HTTP using SOAP. This section describes how to consume this service. The first step is to open Visual Studio 2008 and create a new ASP.NET application. Although we are using an ASP.NET application, you can make this consumer call through any other application type within .NET as well.

Call the new ASP.NET application `WCFConsumer`. This application consumes the `Calculator` service, so it should be laid out with two text boxes and a button to initiate the service call. For this example, we will only use the `Add()` method of the service.

Adding a Service Reference

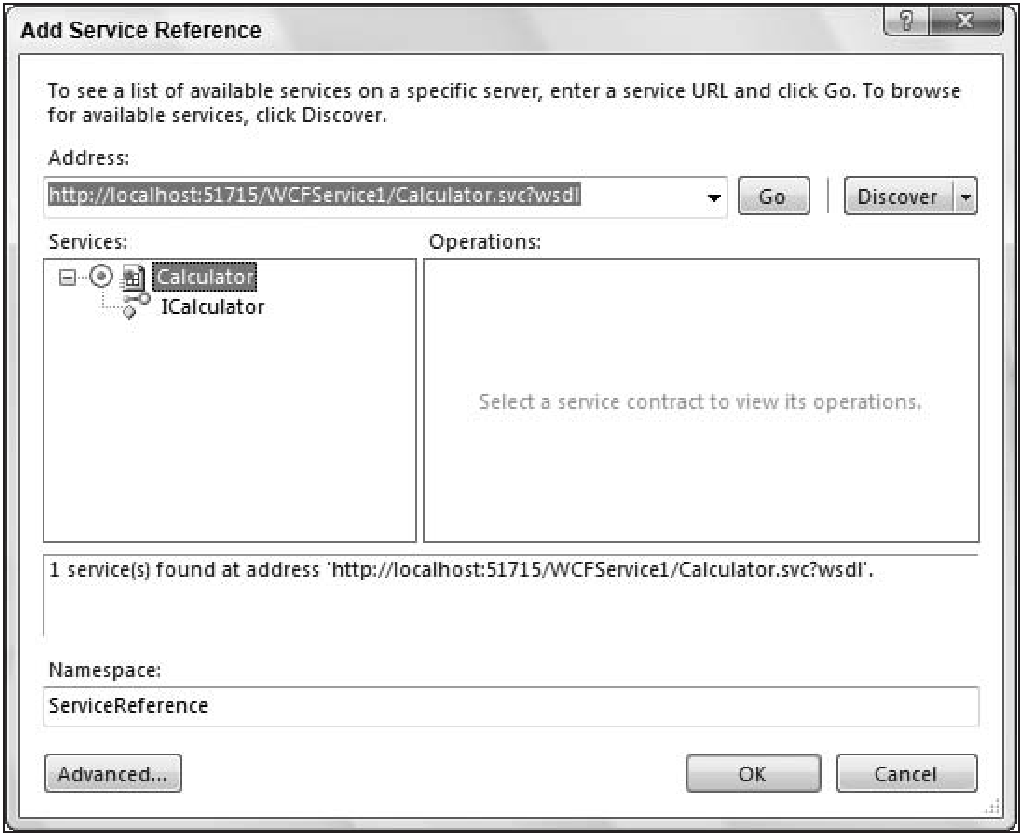
Once you have laid out your ASP.NET page, make a reference to the new WCF service. You do this in a manner quite similar to how it is done with XML Web service references. Right-click on the solution name from the Solution Explorer in Visual Studio and select Add Service Reference from the dialog box. This capability to add a service reference is new to Visual Studio 2008 — previously, you had only the Add Reference and Add Web Reference options.

Once you have selected Add Service Reference, you are presented with the dialog box shown in Figure 29-23.

The Add Service Reference dialog box asks you for two things: the Service URI or Address (basically a pointer to the WSDL file) and the name you want to give to the reference. The name you provide the reference is the name that will be used for the instantiated object that enables you to interact with the service.

Referring to Figure 29-23, you can see that the name provided to the Service Address setting is what is used for the running service from earlier in this chapter. Press OK in the Add Service Reference dialog box. This adds to your project a Service References folder containing some proxy files, as shown in Figure 29-24.

Indeed, the Service References folder is added and two files are contained within this folder: `CalculatorService.map` and `CalculatorService.vb`. The other important addition to note is the `System.ServiceModel` reference, made for you in the References folder. This reference was not there before you made reference to the service through the Add Service Reference dialog.


Add Service Reference
Go
Discover
Advanced...
OK
Cancel

To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

Address: http://localhost:51715/WCFService1/Calculator.svc?wsdl

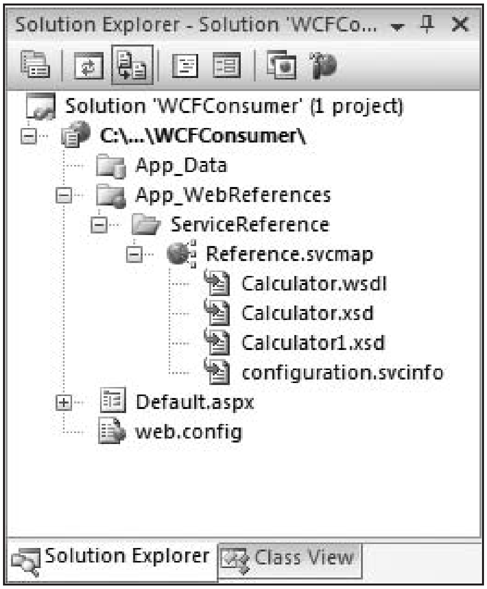
Services:
- Calculator
  - ICalculator

Operations: Select a service contract to view its operations.

1 service(s) found at address 'http://localhost:51715/WCFService1/Calculator.svc?wsdl'.

Namespace: ServiceReference

Figure 29-23


Solution Explorer - Solution 'WCFCo...
Solution Explorer
Class View

- Solution 'WCFConsumer' (1 project)
  - C:\...\WCFConsumer\
    - App_Data
    - App_WebReferences
      - ServiceReference
        - Reference.svcmap
          - Calculator.wsdl
          - Calculator.xsd
          - Calculator1.xsd
          - configuration.svcinfo
    - Default.aspx
    - web.config

Figure 29-24

Configuration File Changes

Looking at the `web.config` file, you can see that Visual Studio has placed information about the service inside the document, as illustrated in Listing 29-30.

Listing 29-30: Additions made to the web.config file by Visual Studio

```
<system.serviceModel>
   <bindings>
      <wsHttpBinding>
         <binding name="WSHttpBinding_ICalculator" closeTimeout="00:01:00"
          openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
          bypassProxyOnLocal="false" transactionFlow="false"
          hostNameComparisonMode="StrongWildcard" maxBufferPoolSize="524288"
          maxReceivedMessageSize="65536" messageEncoding="Text"
          textEncoding="utf-8" useDefaultWebProxy="true" allowCookies="false">
            <readerQuotas maxDepth="32" maxStringContentLength="8192"
             maxArrayLength="16384" maxBytesPerRead="4096"
             maxNameTableCharCount="16384"/>
            <reliableSession ordered="true" inactivityTimeout="00:10:00"
             enabled="false"/>
            <security mode="Message">
               <transport clientCredentialType="Windows" proxyCredentialType="None"
                realm=""/>
               <message clientCredentialType="Windows"
                negotiateServiceCredential="true" algorithmSuite="Default"
                establishSecurityContext="true"/>
            </security>
         </binding>
      </wsHttpBinding>
   </bindings>
   <client>
      <endpoint address="http://localhost:51715/WCFService1/Calculator.svc"
       binding="wsHttpBinding" bindingConfiguration="WSHttpBinding_ICalculator"
       contract="ServiceReference.ICalculator" name="WSHttpBinding_ICalculator">
         <identity>
            <userPrincipalName value="Lipper-STL-LAP\Bill"/>
         </identity>
      </endpoint>
   </client>
</system.serviceModel></ServiceReference>
```

The important part of this configuration document is the `<client>` element. This element contains a child element called `<endpoint>` that defines the *where* and *how* of the service consumption process.

The `<endpoint>` element provides the address of the service — `http://localhost:51715/WCFService1/Calculator.svc` — and it specifies which binding of the available WCF bindings should be used. In this case, the `wsHttpBinding` is the required binding. Even though you are using an established binding from the WCF framework, from the client side you can customize how this binding behaves. The settings that define the behavior of the binding are specified using the `bindingConfiguration` attribute of the `<endpoint>` element. In this case, the value provided to the `bindingConfiguration` attribute is `WSHttpBinding_ICalculator`, which is a reference to the `<binding>` element contained within the `<wsHttpBinding>` element:

As demonstrated, the Visual Studio 2008 enhancements for WCF do make the consumption of these services fairly trivial. The next step is to code the consumption of the service interface to the GUI that was created as one of the first steps.

Writing the Consumption Code

The code to consume the interface is quite minimal. The end user places a number in each of the two text boxes provided and clicks the button to call the service to perform the designated operation on the provided numbers. Listing 29-31 is the code from the button click event:

Listing 29-31: The button click event to call the service

VB

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
  Handles Button1.Click

  Dim ws As New ServiceReference.CalculatorClient()
  Dim result As Integer

   result = ws.Add(TextBox1.Text, TextBox2.Text)
   Response.Write(result.ToString())

   ws.Close()
End Sub
```

C#

```
protected void Button1_Click(object sender, EventArgs e)
{
   ServiceReference.CalculatorClient ws = new ServiceReference.CalculatorClient();

   int result = ws.Add(int.Parse(TextBox1.Text), int.Parse(TextBox2.Text));
   Response.Write(result);

   ws.Close();
}
```

This is quite similar to what is done when working with Web references from the XML Web services world. First is an instantiation of the proxy class, as shown with the creation of the `svc` object:

```
Dim ws As New ServiceReference.CalculatorClient()
```

Working with the `ws` object now, the IntelliSense options provide you with the appropriate `Add()`, `Subtract()`, `Multiply()`, and `Divide()` methods.

As before, the requests and responses are sent over HTTP as SOAP 1.2. This concludes the short tutorial demonstrating how to build your own WCF service using the HTTP protocol and consume this service directly into an ASP.NET application.

Working with Data Contracts

Thus far, when building the WCF services, the defined data contract has relied upon simple types or primitive datatypes. In the case of the earlier WCF service, a .NET type of `Integer` was exposed, which in turn was mapped to an XSD type of `int`. While you may not be able to see the input and output types defined in the WSDL document provided via the WCF-generated one, they are there. These types are actually exposed through an imported `.xsd` document (`Calculator.xsd`and `Calculator1.xsd`). This bit of the WSDL document is presented in Listing 29-32:

Listing 29-32: Imported types in the WSDL document

```
<wsdl:types>
   <xsd:schema targetNamespace="http://tempuri.org/Imports">
      <xsd:import
       schemaLocation="http://localhost:51715/WCFService1/Calculator.svc?xsd=xsd0"
       namespace="http://tempuri.org/" />
      <xsd:import
       schemaLocation="http://localhost:51715/WCFService1/Calculator.svc?xsd=xsd1"
       namespace="http://schemas.microsoft.com/2003/10/Serialization/" />
   </xsd:schema>
</wsdl:types>
```

Typing in the XSD location of `http://localhost:51715/WCFService1/Calculator.svc?xsd=xsd0` gives you the input and output parameters of the service. For instance, looking at the definition of the `Add()` method, you will see the following bit of code, as shown in Listing 29-33.

Listing 29-33: Defining the required types in the XSD

```
<xs:element name="Add">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="a" type="xs:int" />
      <xs:element minOccurs="0" name="b" type="xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="AddResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="AddResult" type="xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

This bit of XML code indicates that there are two required input parameters (`a` and `b`) that are of type `int`; in return, the consumer gets an element called `<AddResult>`, which contains a value of type `int`.

As a builder of this WCF service, you did not have to build the data contract because this service used simple types. When using complex types, you have to create a data contract in addition to your service contract.

Building a Service with a Data Contract

For an example of working with data contracts, create a new WCF service (another WCF service project) called `WCF_WithDataContract`. In this case, you still need an interface that defines your service contract, and then another class that implements that interface. In addition to these items, you also need another class that defines the data contract.

As with the service contract, which makes use of the `<ServiceContract()>` and the `<Operation-Contract()>` attributes, the data contract uses the `<DataContract()>` and `<DataMember()>` attributes. To gain access to these attributes, you have to make a reference to the `System.Runtime.Serialization` namespace in your project and import this namespace into the file.

The custom type in this case is a `Customer` type, as presented in Listing 29-34.

Listing 29-34: Building the Customer type

VB

```
Imports System
Imports System.ServiceModel
Imports System.Runtime.Serialization

<DataContract()> _
Public Class Customer
    <DataMember()> _
    Public FirstName As String

    <DataMember()> _
    Public LastName As String
End Class

<ServiceContract()> _
Public Interface IHelloCustomer
    <OperationContract()> _
    Function HelloFirstName(ByVal cust As Customer) As String

    <OperationContract()> _
    Function HelloFullName(ByVal cust As Customer) As String
End Interface
```

C#

```
using System.Runtime.Serialization;
using System.ServiceModel;

[DataContract]
public class Customer
{
    [DataMember]
    public string Firstname;

    [DataMember]
    public string Lastname;
}
```

Continued

```
[ServiceContract] IHelloCustomer
public interface IWithContract
{
    [OperationContract]
    string HelloFirstName(Customer cust);

    [OperationContract]
    string HelloFullName(Customer cust);
}
```

Here, you can see that the `System.Runtime.Serialization` namespace is also imported, and the first class in the file is the data contract of the service. This class, the `Customer` class, has two members: `FirstName` and `LastName`. Both of these properties are of type `String`. You specify a class as a data contract through the use of the `<DataContract()>` attribute:

```
<DataContract()> _
Public Class Customer

   ' Code removed for clarity

End Class
```

Now, any of the properties contained in the class are also part of the data contract through the use of the `<DataMember()>` attribute:

```
<DataContract()> _
Public Class Customer
    <DataMember()> _
    Public FirstName As String

    <DataMember()> _
    Public LastName As String
End Class
```

Finally, the `Customer` object is used in the interface, as well as the class that implements the `IHelloCustomer` interface, as shown in Listing 29-35.

Listing 29-35: Implementing the interface

VB

```
Public Class HelloCustomer
    Implements IHelloCustomer

    Public Function HelloFirstName(ByVal cust As Customer) As String _
      Implements IHelloCustomer.HelloFirstName
        Return "Hello " & cust.FirstName
    End Function

    Public Function HelloFullName(ByVal cust As Customer) As String _
      Implements IHelloCustomer.HelloFullName
        Return "Hello " & cust.FirstName & " " & cust.LastName
    End Function
End Class
```

C#

```
public class HelloCustomer: IHelloCustomer
{
    public string HelloFirstName(Customer cust)
    {
        return "Hello " + cust.Firstname;
    }

    public string HelloFullName(Customer cust)
    {
        return "Hello " + cust.Firstname + " " + cust.Lastname;
    }
}
```

Building the Consumer

Now that the service is running and in place, the next step is to build the consumer. To begin, build a new ASP.NET application from Visual Studio 2008 and call the project `HelloWorldConsumer`. Again, right-click on the solution and select Add Service Reference from the options provided in the menu.

From the Add Service Reference dialog box, add the location of the WSDL file for the service and press OK. This adds the changes to the references and the `web.config` file just as before, enabling you to consume the service. The following code, as presented in Listing 29-36, shows what is required to consume the service if you are using a Button control to initiate the call.

Listing 29-36: Consuming a custom type through a WCF service

VB

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
  Handles Button1.Click

   Dim ws As New ServiceReference.HelloCustomerClient()
   Dim myCustomer As New ServiceReference.Customer()

   myCustomer.Firstname = "Bill"
   myCustomer.Lastname = "Evjen"

   Response.Write(ws.HelloFullName(myCustomer))

   ws.Close()
End Sub
```

C#

```
protected void Button1_Click(object sender, EventArgs e)
{
   ServiceReference.HelloCustomerClient ws = new
      ServiceReference.HelloCustomerClient();
   ServiceReference.Customer myCustomer = new ServiceReference.Customer();
```

Continued

```
      myCustomer.Firstname = "Bill";
      myCustomer.Lastname = "Evjen";

      Response.Write(ws.HelloFullName(myCustomer));

      ws.Close();
   }
```

As a consumer, once you make the reference, you will notice that the service reference provides both a `HelloCustomerClient` object andthe `Customer` object, which was defined through the service's data contract.

Therefore, the preceding code block just instantiates both of these objects and builds the `Customer` object before it is passed into the `HelloFullName()` method provided by the service.

Looking at WSDL and the Schema for HelloCustomerService

When you make a reference to the `HelloCustomer` service, you will find the following XSD imports in the WSDL:

```
<wsdl:types>
   <xsd:schema targetNamespace="http://tempuri.org/Imports">
    <xsd:import
     schemaLocation="http://localhost:51715/WCFService1/HelloCustomer.svc?xsd=xsd0"
     namespace="http://tempuri.org/" />
    <xsd:import
     schemaLocation="http://localhost:51715/WCFService1/HelloCustomer.svc?xsd=xsd1"
     namespace="http://schemas.microsoft.com/2003/10/Serialization/" />
    <xsd:import
     schemaLocation="http://localhost:51715/WCFService1/HelloCustomer.svc?xsd=xsd2"
     namespace="http://schemas.datacontract.org/2004/07/" />
   </xsd:schema>
</wsdl:types>
```

`http://localhost:51715/WCFService1/HelloCustomer.svc?xsd=xsd2` provides the details on your `Customer` object. The code from this file is shown here:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema elementFormDefault="qualified"
 targetNamespace="http://schemas.datacontract.org/2004/07/"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://schemas.datacontract.org/2004/07/">
   <xs:complexType name="Customer">
      <xs:sequence>
         <xs:element minOccurs="0" name="Firstname" nillable="true"
          type="xs:string" />
         <xs:element minOccurs="0" name="Lastname" nillable="true"
          type="xs:string" />
      </xs:sequence>
   </xs:complexType>
   <xs:element name="Customer" nillable="true" type="tns:Customer" />
</xs:schema>
```

This is an XSD description of the `Customer` object. Making a reference to the WSDL includes the XSD description of the `Customer` object and gives you the ability to create local instances of this object.

Using this model, you can easily build your services with your own defined types.

Namespaces

Note that the services built in the chapter have no defined namespaces. If you looked at the WSDL files that were produced, you would see that the namespace provided is `http://tempuri.org`. Obviously, you do not want to go live with this default namespace. Instead, you need to define your own namespace.

To accomplish this task, the interface's `<ServiceContract()>` attribute enables you to set the namespace, as shown here:

```
<ServiceContract(Namespace:="http://www.lipperweb.com/ns/")> _
Public Interface IHelloCustomer
    <OperationContract()> _
    Function HelloFirstName(ByVal cust As Customer) As String

    <OperationContract()> _
    Function HelloFullName(ByVal cust As Customer) As String
End Interface
```

Here, the `<ServiceContract()>` attribute uses the `Namespace` property to provide a namespace.

Summary

This chapter was a whirlwind tour of XML Web services in the .NET platform. It is definitely a topic that merits an entire book of its own. The chapter showed you the power of exposing your data and logic as SOAP and also how to consume these SOAP messages directly in the ASP.NET applications you build.

In addition to pointing out the power you have for building and consuming basic Web services, the chapter spent some time helping you understand caching, performance, the use of SOAP headers, and more. A lot of power is built into this model; every day the Web services model is starting to make stronger inroads into various enterprise organizations. It is becoming more likely that to get at some data or logic you need for your application, you will employ the tactics presented here.

This chapter also looked at one of the newest capabilities provided to the .NET Framework world. The .NET Framework 3.5 and WCF are a great combination for building advanced services that take ASP.NET Web services, .NET Remoting, Enterprise Services, and MSMQ to the next level.

While not exhaustive, this chapter broadly outlined the basics of the framework. As you start to dig deeper in the technology, you will find capabilities that are strong and extensible.

30

Localization

Developers usually build Web applications in their native language and then, as the audience for the application expands, they then realize the need to globalize the application. Of course, the ideal is to build the Web application to handle an international audience right from the start — but, in many cases, this may not be possible because of the extra work it requires.

It is good to note that with the ASP.NET 3.5 framework, a considerable effort has been made to address the internationalization of Web applications. You quickly realize that changes to the API, the addition of capabilities to the server controls, and even Visual Studio itself equip you to do the extra work required more easily to bring your application to an international audience. This chapter looks at some of the important items to consider when building your Web applications for the world.

Cultures and Regions

The ASP.NET page that is pulled up in an end user's browser runs under a specific culture and region setting. When building an ASP.NET application or page, the defined culture in which it runs is dependent upon both a culture and region setting coming from the server in which the application is run or from a setting applied by the client (the end user). By default, ASP.NET runs under a culture setting defined by the server.

The world is made up of a multitude of cultures, each of which has a language and a set of defined ways in which it views and consumes numbers, uses currencies, sorts alphabetically, and so on. The .NET Framework defines cultures and regions using the *Request for Comments 1766* standard definition (tags for identification of languages) that specifies a language and region using two-letter codes separated by a dash. The following table provides examples of some culture definitions.

Culture Code	Description
en-US	English language; United States
en-GB	English language; United Kingdom (Great Britain)
en-AU	English language; Australia
en-CA	English language; Canada

Looking at the examples in this table, you can see that four distinct cultures are defined. These four cultures have some similarities and some differences. All four cultures speak the same language (English). For this reason, the language code of `en` is used in each culture setting. After the language setting comes the region setting. Even though these cultures speak the same language, it is important to distinguish them further by setting their region (such as `US` for the United States, `GB` for the United Kingdom, `AU` for Australia, and `CA` for Canada). As you are probably well aware, the English language in the United States is slightly different from the English language that is used in the United Kingdom, and so forth. Beyond language, differences exist in how dates and numerical values are represented. This is why a culture's language and region are presented together.

The differences do not break down by the country only. Many times, countries contain more than a single language and each area has its own preference for notation of dates and other items. For example, `en-CA` specifies English speakers in Canada. Because Canada is not only an English-speaking country, it also includes the culture setting of `fr-CA` for French-speaking Canadians.

Understanding Culture Types

The culture definition you have just seen is called a *specific culture* definition. This definition is as detailed as you can possibly get — defining both the language and the region. The other type of culture definition is a *neutral culture* definition. Each specific culture has a specified neutral culture that it is associated with. For instance, the English language cultures shown in the previous table are separate, but they also all belong to one neutral culture EN (English). The diagram presented in Figure 30-1 displays how these culture types relate to one another.

From this diagram, you can see that many specific cultures belong to a neutral culture. Higher in the hierarchy than the neutral culture is an *invariant culture*, which is an agnostic culture setting that should be utilized when passing items (such as dates and numbers) around a network. When performing these kinds of operations, make your back-end data flows devoid of user-specific culture settings. Instead, apply these settings in the business and presentation layers of your applications.

Also, pay attention to the neutral culture when working with your applications. Invariably, you are going to build applications with views that are more dependent on a neutral culture rather than on a specific culture. For instance, if you have a Spanish version of your application, you probably make this version available to all Spanish speakers regardless of their regions. In many applications, it will not matter if the Spanish speaker is from Spain, Mexico, or Argentina. In a case where it does make a difference, use the specific culture settings.

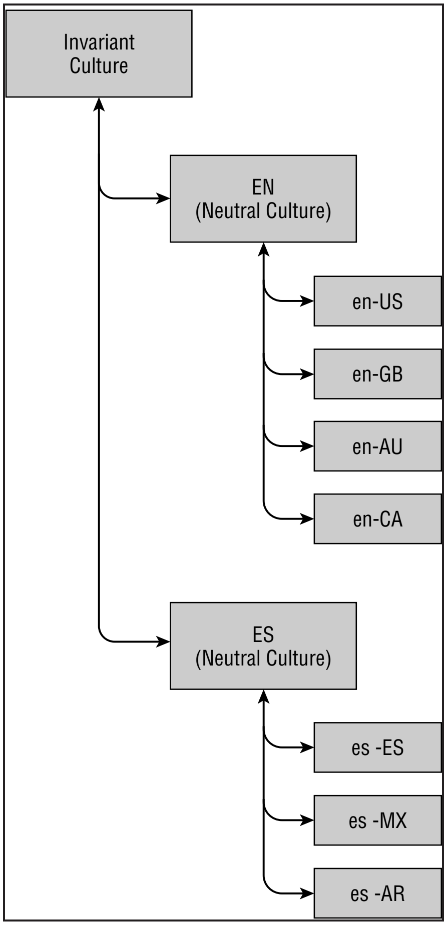

Invariant Culture
EN (Neutral Culture)
en-US
en-GB
en-AU
en-CA
ES (Neutral Culture)
es -ES
es -MX
es -AR

Figure 30-1

The ASP.NET Threads

When the end user requests an ASP.NET page, this Web page is executed on a thread from the thread pool. The thread has a culture associated with it. You can get information about the culture of the thread programmatically and then check for particular details about that culture. This is illustrated in Listing 30-1.

Listing 30-1: Checking the culture of the ASP.NET thread

VB

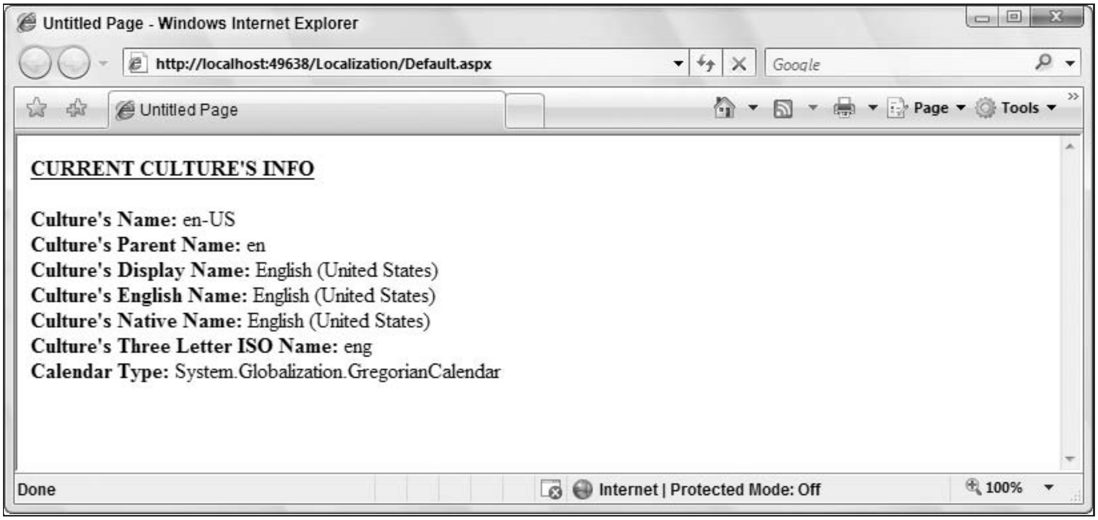
```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
  Dim ci As CultureInfo = System.Threading.Thread.CurrentThread.CurrentCulture
  Response.Write("<b><u>CURRENT CULTURE'S INFO</u></b>")
  Response.Write("<p><b>Culture's Name:</b> " & ci.Name.ToString() & "<br>")
```

Continued

```
   Response.Write("<b>Culture's Parent Name:</b> " & ci.Parent.Name.ToString() & _
      "<br>")
   Response.Write("<b>Culture's Display Name:</b> " & ci.DisplayName.ToString() & _
      "<br>")
   Response.Write("<b>Culture's English Name:</b> " & ci.EnglishName.ToString() & _
      "<br>")
   Response.Write("<b>Culture's Native Name:</b> " & ci.NativeName.ToString() & _
      "<br>")
   Response.Write("<b>Culture's Three Letter ISO Name:</b> " & _
      ci.Parent.ThreeLetterISOLanguageName.ToString() & "<br>")
   Response.Write("<b>Calendar Type:</b> " & ci.Calendar.ToString() & "</p >")
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
   CultureInfo ci = System.Threading.Thread.CurrentThread.CurrentCulture;
   Response.Write("<b><u>CURRENT CULTURE'S INFO</u></b>");
   Response.Write("<p><b>Culture's Name:</b> " + ci.Name.ToString() + "<br>");
   Response.Write("<b>Culture's Parent Name:</b> " + ci.Parent.Name.ToString() +
      "<br>");
   Response.Write("<b>Culture's Display Name:</b> " + ci.DisplayName.ToString() +
      "<br>");
   Response.Write("<b>Culture's English Name:</b> " + ci.EnglishName.ToString() +
      "<br>");
   Response.Write("<b>Culture's Native Name:</b> " + ci.NativeName.ToString() +
      "<br>");
   Response.Write("<b>Culture's Three Letter ISO Name:</b> " +
      ci.Parent.ThreeLetterISOLanguageName.ToString() + "<br>");
   Response.Write("<b>Calendar Type:</b> " + ci.Calendar.ToString() + "</p >");
}
```


Untitled Page - Windows Internet Explorer
http://localhost:49638/Localization/Default.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

```
CURRENT CULTURE'S INFO

Culture's Name: en-US
Culture's Parent Name: en
Culture's Display Name: English (United States)
Culture's English Name: English (United States)
Culture's Native Name: English (United States)
Culture's Three Letter ISO Name: eng
Calendar Type: System.Globalization.GregorianCalendar
```

Figure 30-2

This bit of code in the `Page_Load` event checks the `CurrentCulture` property. You can place the result of this value in a `CultureInfo` object. To get at this object, you import the `System.Globalization` namespace into your Web page. The `CultureInfo` object contains a number of properties that provides you with specific culture information. The following items, which are displayed in a series of simple `Response.Write` statements, are only a small sampling of what is actually available. Running this page produces results similar to what is shown in Figure 30-2.

From this figure, you can see that the en-US culture is the default setting in which the ASP.NET thread executes. In addition to this, you can use the `CultureInfo` object to get at a lot of other descriptive information about the culture.

You can always change a thread's culture on the overloads provided via a new instantiation of the `CultureInfo` object. This is presented in Listing 30-2.

Listing 30-2: Changing the culture of the thread using the CultureInfo object

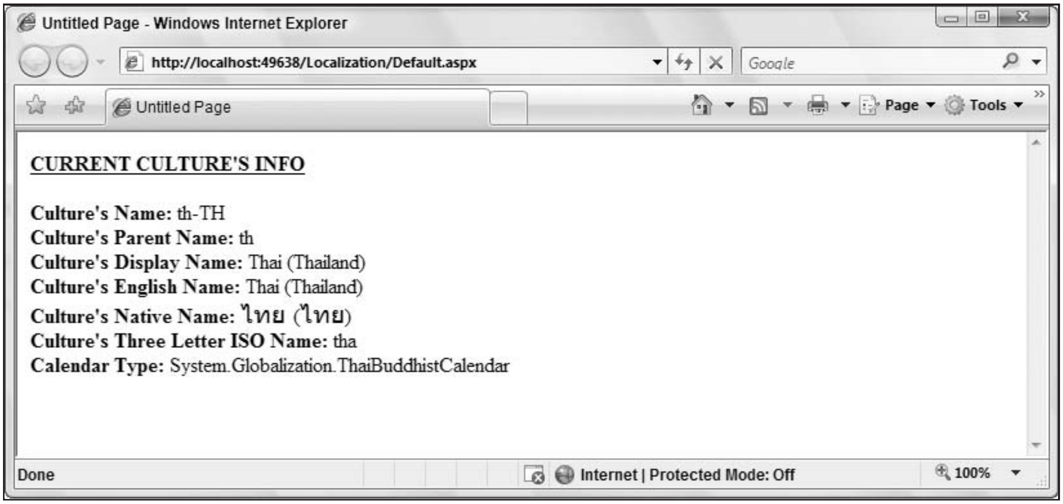
VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   System.Threading.Thread.CurrentThread.CurrentCulture = New CultureInfo("th-TH")
   Dim ci As CultureInfo = System.Threading.Thread.CurrentThread.CurrentCulture
   Response.Write("<b><u>CURRENT CULTURE'S INFO</u></b>")
   Response.Write("<p><b>Culture's Name:</b> " & ci.Name.ToString() & "<br>")
   Response.Write("<b>Culture's Parent Name:</b> " & ci.Parent.Name.ToString() & _
      "<br>")
   Response.Write("<b>Culture's Display Name:</b> " & ci.DisplayName.ToString() & _
      "<br>")
   Response.Write("<b>Culture's English Name:</b> " & ci.EnglishName.ToString() & _
      "<br>")
   Response.Write("<b>Culture's Native Name:</b> " & ci.NativeName.ToString() & _
      "<br>")
   Response.Write("<b>Culture's Three Letter ISO Name:</b> " & _
      ci.Parent.ThreeLetterISOLanguageName.ToString() & "<br>")
   Response.Write("<b>Calendar Type:</b> " & ci.Calendar.ToString() & "</p >")
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
   System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("th-TH");
   CultureInfo ci = System.Threading.Thread.CurrentThread.CurrentCulture;
   Response.Write("<b><u>CURRENT CULTURE'S INFO</u></b>");
   Response.Write("<p><b>Culture's Name:</b> " + ci.Name.ToString() + "<br>");
   Response.Write("<b>Culture's Parent Name:</b> " + ci.Parent.Name.ToString() +
      "<br>");
   Response.Write("<b>Culture's Display Name:</b> " + ci.DisplayName.ToString() +
      "<br>");
   Response.Write("<b>Culture's English Name:</b> " + ci.EnglishName.ToString() +
      "<br>");
   Response.Write("<b>Culture's Native Name:</b> " + ci.NativeName.ToString() +
      "<br>");
   Response.Write("<b>Culture's Three Letter ISO Name:</b> " +
      ci.Parent.ThreeLetterISOLanguageName.ToString() + "<br>");
   Response.Write("<b>Calendar Type:</b> " + ci.Calendar.ToString() + "</p>");
}
```

In this example, only a single line of code is added to assign a new instance of the `CultureInfo` object to the `CurrentCulture` property of the thread being executed by ASP.NET. The culture setting enables the `CultureInfo` object to define the culture you want to utilize. In this case, the Thai language of Thailand is assigned, and the results produced in the browser are illustrated in Figure 30-3.


Untitled Page - Windows Internet Explorer
http://localhost:49638/Localization/Default.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

CURRENT CULTURE'S INFO

Culture's Name: th-TH
Culture's Parent Name: th
Culture's Display Name: Thai (Thailand)
Culture's English Name: Thai (Thailand)
Culture's Native Name: ไทย (ไทย)
Culture's Three Letter ISO Name: tha
Calendar Type: System.Globalization.ThaiBuddhistCalendar

Figure 30-3

From this figure, you can see that the .NET Framework goes so far as to provide the native name of the language used even if it is not a Latin-based letter style. In this case, the results are presented for the Thai language in Thailand, and some of the properties that are associated with this culture (such as an entirely different calendar than is the one used in Western Europe and the United States). Remember that you reference `System.Globalization` to get at the `CultureInfo` object.

Server-Side Culture Declarations

ASP.NET enables you to easily define the culture that is used either by your entire ASP.NET application or by a specific page within your application. You can specify the culture for any of your ASP.NET applications by means of the appropriate configuration files. In the default install of ASP.NET, no culture is specified as is evident when you look at the global `web.config.comments` file (meant for documentation purposes) found in the ASP.NET 2.0 CONFIG folder (`C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG`). Remember that ASP.NET 3.5 is built on top of ASP.NET 2.0 and uses the same configuration files. In the `web.config.comments` file, you find a `<globalization>` section of the configuration document. This section is presented in Listing 30-3.

Listing 30-3: The <globalization> section in the web.config.comments file

```
<globalization requestEncoding="utf-8" responseEncoding="utf-8" fileEncoding=""
  culture="" uiCulture="" enableClientBasedCulture="false"
  responseHeaderEncoding="utf-8" resourceProviderFactoryType=""
  enableBestFitResponseEncoding="false" />
```

Note the two attributes represented in bold — `culture` and `uiCulture`. The `culture` attribute enables you to define the culture to use for processing incoming requests, whereas the `uiCulture` attribute enables you define the default culture needed to process any resource files in the application. (The use of these attributes will be covered later in the chapter.)

As you look at the configuration declaration in Listing 30-3, you can see that nothing is specified for the culture settings. One option you have when specifying a culture on the server is to define this culture in the server version of the `web.config` file found in the CONFIG folder. This causes every ASP.NET 3.5 application on this server to adopt this particular culture setting. The other option is to specify these settings in the `web.config` file of the application as illustrated in Listing 30-4.

Listing 30-4: Defining the <globalization> section in the web.config file

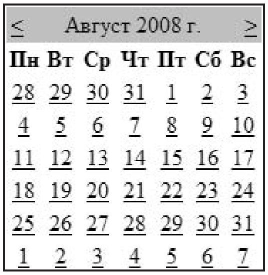
```
<configuration>
   <system.web>

      <globalization culture="ru-RU" uiCulture="ru-RU" />

   </system.web>
</configuration>
```

In this case, the culture established for just this ASP.NET application is the Russian language in the country of Russia. In addition to setting the culture at either the server-wide the application-wide level, another option is to set the culture at the page-level. This is illustrated in Listing 30-5.

Listing 30-5: Defining the culture at the page level @page using the directive

```
<%@ Page Language="VB" UICulture="ru-RU" Culture="ru-RU" %>
```

This example determines that the Russian language and culture settings are used for everything on the page. You can see this in action by using this `@Page` directive and a simple calendar control on the page. Figure 30-4 shows the output.



<			Август 2008 г.			>
Пн	Вт	Ср	Чт	Пт	Сб	Вс
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

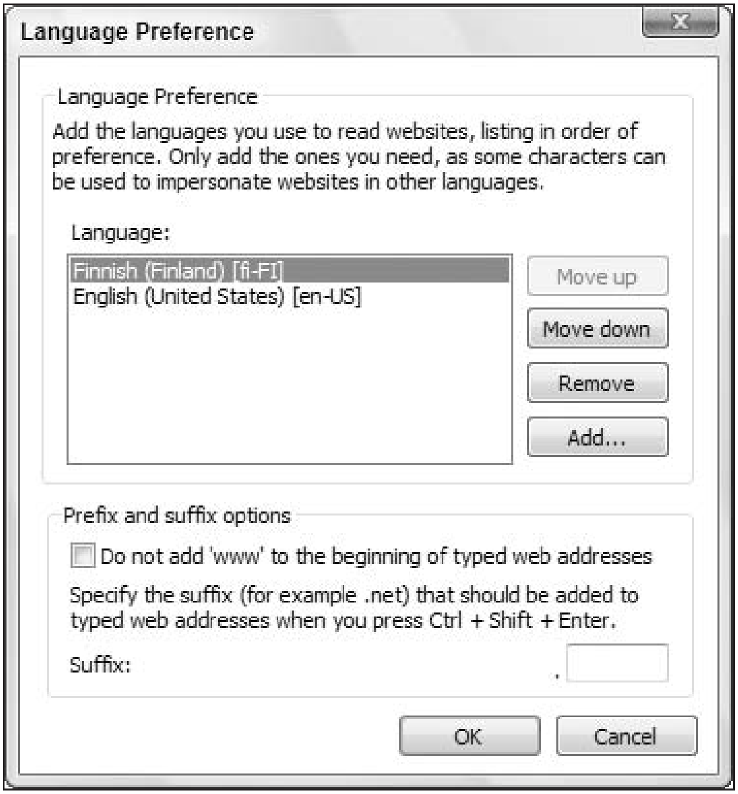
Figure 30-4

Client-Side Culture Declarations

In addition to using server-side setting to define the culture for your ASP.NET pages, you also have the option of defining the culture with what the client has set as his preference in a browser instance.

When end users install Microsoft's Internet Explorer and some of the other browsers, they have the option to select their preferred cultures in a particular order (if they have selected more than a single culture preference). To see this in action in IE, select Tools ➪ Internet Options from the IE menu. On the first tab provided (General), you see a Languages button at the bottom of the dialog. Select this button and you are provided with the Language Preference dialog shown in Figure 30-5.

In this figure, you can see that two cultures are selected from the list of available cultures. To add any additional cultures to the list, select the Add button from the dialog and select the appropriate culture from the list. After you have selected cultures that are present in the list, you can select the order in which you prefer to use them. In the case of Figure 30-5, the Finnish culture is established as the most preferred culture, whereas the U.S. version of English is selected as the second preference. A user with this setting gets the Finnish language version of the application before anything else; if a Finnish version is not available, a U.S. English version is presented.



Add the languages you use to read websites, listing in order of preference. Only add the ones you need, as some characters can be used to impersonate websites in other languages.

Finnish (Finland) [fi-FI]
English (United States) [en-US]

☐ Do not add 'www' to the beginning of typed web addresses

Specify the suffix (for example .net) that should be added to typed web addresses when you press Ctrl + Shift + Enter.

Figure 30-5

After the end user selects, you can use the auto feature provided in ASP.NET 3.5. Instead of specifying a distinct culture in any of the configuration files or from the `@Page` directive, you can also state that ASP.NET should automatically select the culture provided by the end user requesting the page. This is done using the `auto` keyword, as illustrated in Listing 30-6.

Listing 30-6: Changing the culture to the end user's selection

```
<%@ Page Language="VB" UICulture="auto" Culture="auto" %>
```

With this construction in your page, the dates, calendars, and numbers now appear in the preferred culture of the requestor. What happens, however, if you have translated resources in resource files (shown later in the chapter) that depends on a culture specification. What if you only have specific translations and so cannot handle every possible culture that might be returned to your ASP.NET page? In this case, you can specify the auto option with an additional fallback option if ASP.NET cannot find the culture settings of the user (such as culture-specific resource files). This usage is illustrated in Listing 30-7.

Listing 30-7: Providing a fallback culture from the auto option

```
<%@ Page Language="VB" UICulture="auto:en-US" Culture="auto:en-US" %>
```

In this case, the automatic detection is utilized, but if the culture the end user prefers is not present, then en-US is used.

Translating Values and Behaviors

In the process of globalizing your ASP.NET application, you may notice a number of items that are done differently than building an application that is devoid of globalization, including how dates are represented and currencies are shown. This next section touches upon some of these topics.

Understanding Differences in Dates

Different cultures specify dates and time very differently. For instance, take the following date as an example:

```
08/11/2008
```

What is this date exactly? Is it August 11, 2008 or is it November 8, 2008I repeat: When storing values such as date/time stamps in a database or other some type of back-end system, you should always use the same culture (or invariant culture) for these items so that you avoid any mistakes. It should be the job of the business logic layer or the presentation layer to convert these items for use by the end user.

Setting the culture at the server-level or in the `@Page` directive (as was discussed earlier) enables ASP.NET to make these conversions for you. You can also simply assign a new culture to the thread in which ASP.NET is running. For instance, look at the code listing presented in Listing 30-8.

Listing 30-8: Working with date/time values in different cultures

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
  Dim dt As DateTime = New DateTime(2008, 8, 11, 11, 12, 10, 10)
  System.Threading.Thread.CurrentThread.CurrentCulture = New CultureInfo("en-US")
  Response.Write("<b><u>en-US</u></b><br>")
  Response.Write(dt.ToString() & "<br>")

  System.Threading.Thread.CurrentThread.CurrentCulture = New CultureInfo("ru-RU")
  Response.Write("<b><u>ru-RU</u></b><br>")
  Response.Write(dt.ToString() & "<br>")

  System.Threading.Thread.CurrentThread.CurrentCulture = New CultureInfo("fi-FI")
```

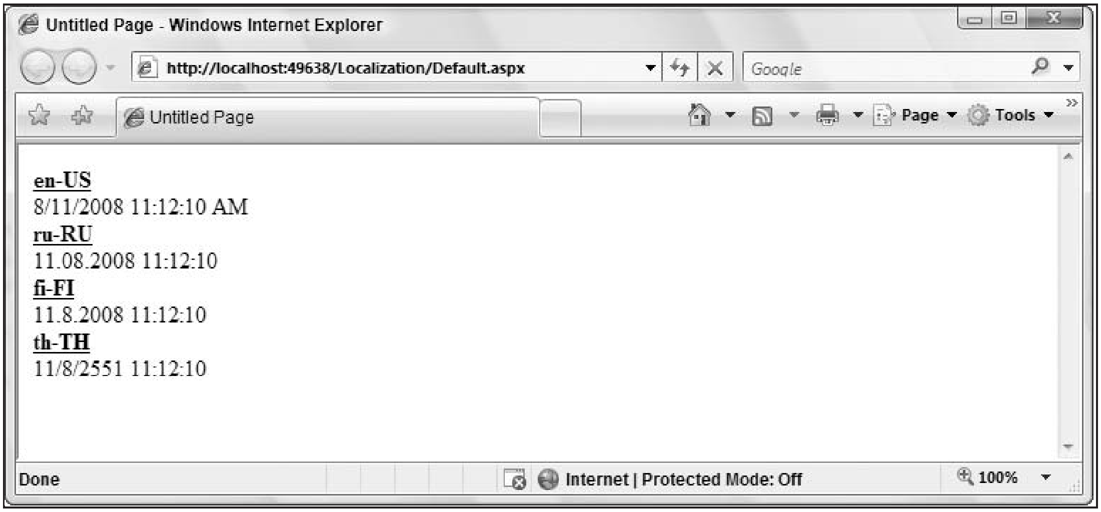
Continued

```
    Response.Write("<b><u>fi-FI</u></b><br>")
    Response.Write(dt.ToString() & "<br>")

    System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("th-TH")
    Response.Write("<b><u>th-TH</u></b><br>")
    Response.Write(dt.ToString())
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    DateTime dt = new DateTime(2008, 8, 11, 11, 12, 10, 10);
    System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
    Response.Write("<b><u>en-US</u></b><br>");
    Response.Write(dt.ToString() + "<br>");

    System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("ru-RU");
    Response.Write("<b><u>ru-RU</u></b><br>");
    Response.Write(dt.ToString() + "<br>");

    System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("fi-FI");
    Response.Write("<b><u>fi-FI</u></b><br>");
    Response.Write(dt.ToString() + "<br>");

    System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("th-TH");
    Response.Write("<b><u>th-TH</u></b><br>");
    Response.Write(dt.ToString());
}
```

In this case, four different cultures are utilized, and the date/time construction used by that culture is written to the browser screen using a `Response.Write` command. The result from this code operation is presented in Figure 30-6.


Untitled Page - Windows Internet Explorer
http://localhost:49638/Localization/Default.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

en-US
8/11/2008 11:12:10 AM
ru-RU
11.08.2008 11:12:10
fi-FI
11.8.2008 11:12:10
th-TH
11/8/2551 11:12:10

Figure 30-6

As you can see, the formats used to represent a date/time value are dramatically different from one another — and one of the cultures, the Thai culture (th-TH), even uses an entirely different calendar that labels this year as 2551.

Understanding Differences in Numbers and Currencies

In addition to date/time values, numbers are constructed quite differently from one culture to the next. How can a number be represented differently in different cultures? Well, it has less to do with the actual number (although certain cultures use different number symbols), and more to do with how the number separators are used for decimals or for showing amounts such as thousands, millions, and more. For instance, in the English culture of the United States (en-US), you see numbers represented in the following fashion:

```
5,123,456.00
```

From this example, you can see that the en-US culture uses a comma as a separator for thousands and a period for signifying the start of any decimals that might appear after the number is presented. It is quite different when working with other cultures. Listing 30-9 shows you an example of representing numbers in other cultures.

Listing 30-9: Working with numbers in different cultures

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   Dim myNumber As Double = 5123456.00
   System.Threading.Thread.CurrentThread.CurrentCulture = New CultureInfo("en-US")
   Response.Write("<b><u>en-US</u></b><br>")
   Response.Write(myNumber.ToString("n") & "<br>")

   System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("vi-VN")
   Response.Write("<b><u>vi-VN</u></b><br>")
   Response.Write(myNumber.ToString("n") & "<br>")

   System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("fi-FI")
   Response.Write("<b><u>fi-FI</u></b><br>")
   Response.Write(myNumber.ToString("n") & "<br>")

   System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("fr-CH")
   Response.Write("<b><u>fr-CH</u></b><br>")
   Response.Write(myNumber.ToString("n"))
End Sub
```

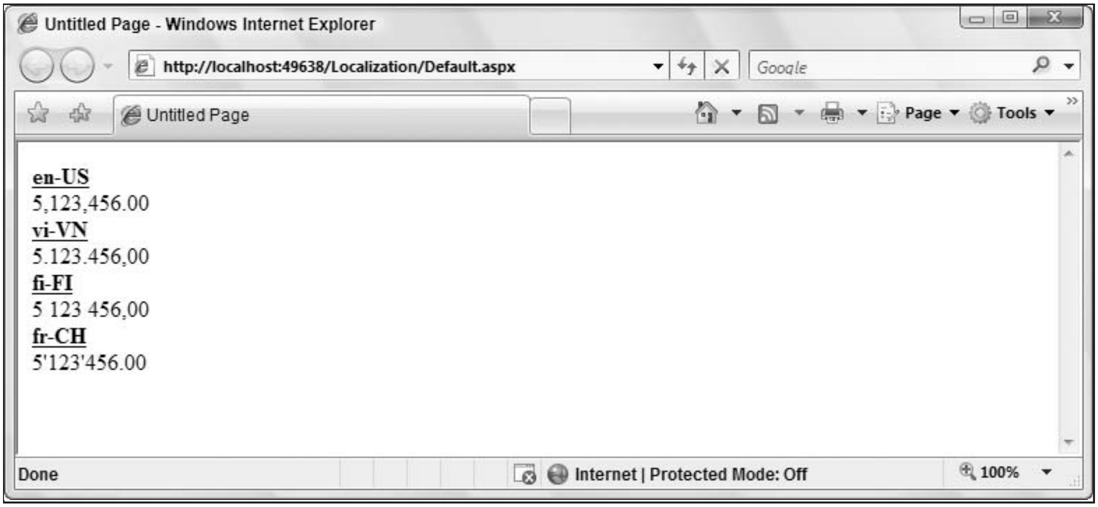
C#

```
protected void Page_Load(object sender, EventArgs e)
{
   double myNumber = 5123456.00;
   System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
   Response.Write("<b><u>en-US</u></b><br>");
   Response.Write(myNumber.ToString("n") + "<br>");

   System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("vi-VN");
   Response.Write("<b><u>vi-VN</u></b><br>");
```

Continued

```
        Response.Write(myNumber.ToString("n") + "<br>");

        System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("fi-FI");
        Response.Write("<b><u>fi-FI</u></b><br>");
        Response.Write(myNumber.ToString("n") + "<br>");

        System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("fr-CH");
        Response.Write("<b><u>fr-CH</u></b><br>");
        Response.Write(myNumber.ToString("n"));
    }
```

Running this short example produces the results presented in Figure 30-7.


Untitled Page - Windows Internet Explorer
http://localhost:49638/Localization/Default.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

```
en-US
5,123,456.00
vi-VN
5.123.456,00
fi-FI
5 123 456,00
fr-CH
5'123'456.00
```

Figure 30-7

From this example, you can see that the other cultures represented here show numbers in quite a different format than that of the en-US culture. The second culture listed in the figure, vi-VN (Vietnamese in Vietnam), constructs a number exactly the opposite from the way it is constructed in en-US. The Vietnamese culture uses periods for the thousand separators and a comma for signifying decimals. Finnish, on the other hand, uses spaces for the thousand separators and a comma for the decimal separator, whereas the French-speaking Swiss use a high comma for separating thousands and a period for the decimal separator. As you can see, it is important to ‘‘translate’’ numbers to the proper construction so that users of your application can properly understand the numbers represented.

You also represent numbers is when working with currencies. It is one thing to *convert* currencies so that end users understand the proper value of an item, but it is another to translate the construction of the currency just as you would a basic number.

Each culture has a distinct currency symbol used to signify that a number represented is an actual currency value. For instance, the en-US culture represents a currency in the following format:

```
$5,123,456.00
```

The en-US culture uses a U.S. Dollar symbol ($), and the location of this symbol is just as important as the symbol itself. For en-US, the $ symbol directly preceeds the currency value (with no space in between the symbol and the first character of the number). Other cultures use different symbols to represent a currency and often place those currency symbols in different locations. Change the previous Listing 30-9 so that it now represents the number as a currency. The necessary changes are presented in Listing 30-10.

Listing 30-10: Working with currencies in different cultures

VB

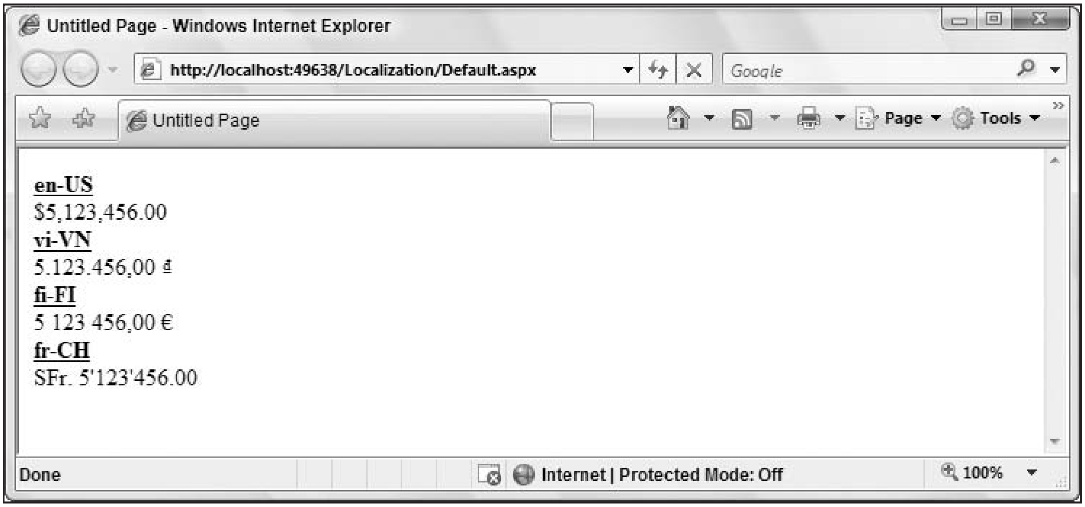
```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   Dim myNumber As Double = 5123456.00
   System.Threading.Thread.CurrentThread.CurrentCulture = New CultureInfo("en-US")
   Response.Write("<b><u>en-US</u></b><br>")
   Response.Write(myNumber.ToString("c") & "<br>")

   System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("vi-VN")
   Response.Write("<b><u>vi-VN</u></b><br>")
   Response.Write(myNumber.ToString("c") & "<br>")

   System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("fi-FI")
   Response.Write("<b><u>fi-FI</u></b><br>")
   Response.Write(myNumber.ToString("c") & "<br>")

   System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("fr-CH")
   Response.Write("<b><u>fr-CH</u></b><br>")
   Response.Write(myNumber.ToString("c"))
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
   double myNumber = 5123456.00;
   System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
   Response.Write("<b><u>en-US</u></b><br>");
   Response.Write(myNumber.ToString("c") + "<br>");

   System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("vi-VN");
   Response.Write("<b><u>vi-VN</u></b><br>");
   Response.Write(myNumber.ToString("c") + "<br>");

   System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("fi-FI");
   Response.Write("<b><u>fi-FI</u></b><br>");
   Response.Write(myNumber.ToString("c") + "<br>");

   System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("fr-CH");
   Response.Write("<b><u>fr-CH</u></b><br>");
   Response.Write(myNumber.ToString("c"));
}
```

Run this example to see how these cultures represent currency values, as illustrated in Figure 30-8.

From this figure, you can see that not only are the numbers constructed quite differently from one another, but the currency symbol and the location of the symbol in regard to the number are quite different as well.


Untitled Page - Windows Internet Explorer
http://localhost:49638/Localization/Default.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

en-US
$5,123,456.00
vi-VN
5.123.456,00 ₫
fi-FI
5 123 456,00 €
fr-CH
SFr. 5'123'456.00

Figure 30-8

When working with currencies, note that when you are using currencies on an ASP.NET page, you have provided an automatic culture setting for the page as a whole (such as setting the culture in the `@Page` directive). You must specify a specific culture for the currency that is the same in all cases *unless* you are actually doing a currency conversion. For instance, if you are specifying a U.S. Dollar currency value on your ASP.NET page, you do not want to specify that the culture of the currency is something else (for example, the Euro). An exception would be if you actually performed a currency conversion and showed the appropriate Euro value along with the culture specification of the currency. Therefore, if you are using an automatic culture setting on your ASP.NET page and you are *not* converting the currency, you perform something similar to what is illustrated in Listing 30-11 for currency values.

Listing 30-11: Reverting to a specific culture when displaying currencies

VB

```
Dim myNumber As Double = 5123456.00
Dim usCurr As CultureInfo = New CultureInfo("en-US")
Response.Write(myNumber.ToString("c", usCurr))
```

C#

```
double myNumber = 5123456.00;
CultureInfo usCurr = new CultureInfo("en-US");
Response.Write(myNumber.ToString("c", usCurr));
```

Understanding Differences in Sorting Strings

You have learned to translate textual values and alter the construction of the numbers, date/time values, currencies, and more when you are globalizing an application. You should also take note when applying culture settings to some of the programmatic behaviors that you establish for values in your applications. One operation that can change based upon the culture setting applied is how .NET sorts strings. You might think that all cultures sort strings in the same way (and generally they do), but sometimes differences exist in how sorting occurs. To give you an example, Listing 30-12 shows you a sorting operation occurring in the en-US culture.

Listing 30-12: Working with sorting in different cultures

VB

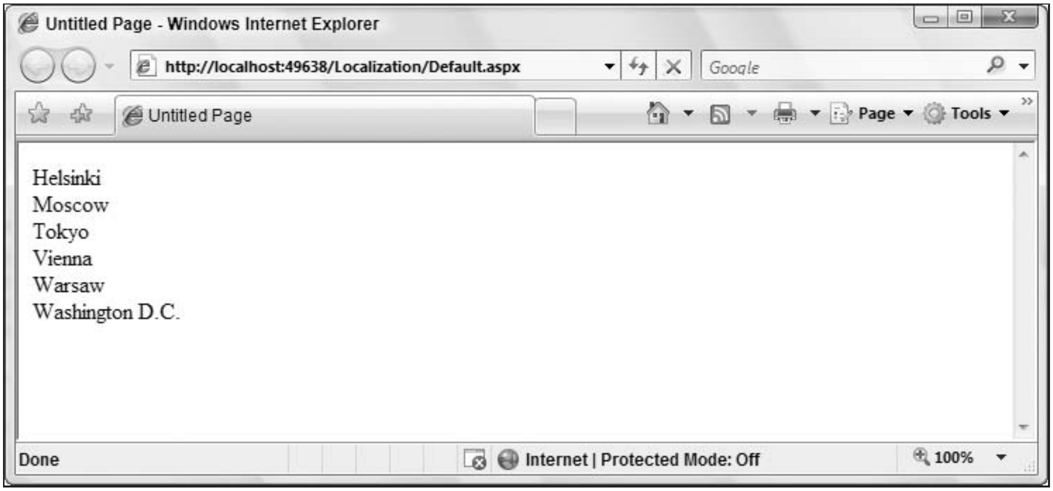
```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   System.Threading.Thread.CurrentThread.CurrentCulture = New CultureInfo("en-US")

   Dim myList As List(Of String) = New List(Of String)

   myList.Add("Washington D.C.")
   myList.Add("Helsinki")
   myList.Add("Moscow")
   myList.Add("Warsaw")
   myList.Add("Vienna")
   myList.Add("Tokyo")

   myList.Sort()

   For Each item As String In myList
      Response.Write(item.ToString() + "<br>")
   Next
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
   System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");

   List<string> myList = new List<string>();
   myList.Add("Washington D.C.");
   myList.Add("Helsinki");
   myList.Add("Moscow");
   myList.Add("Warsaw");
   myList.Add("Vienna");
   myList.Add("Tokyo");

   myList.Sort();

   foreach (string item in myList)
   {
      Response.Write(item.ToString() + "<br>");
   }
}
```

For this example to work, you have to import the `System.Collections` and the `System.Collections.Generic` namespaces because this example makes use of the `List(Of String)` object.

In this example, a generic list of capitals from various countries of the world is created in random order. Then the `Sort()` method of the generic `List(Of String)` object is invoked. This sorting operation sorts the strings based upon how sorting is done for the defined culture in which the ASP.NET thread is running. Listing 30-12 shows the sorting as it is done for the en-US culture. The result of this operation is presented in Figure 30-9.



Untitled Page - Windows Internet Explorer
http://localhost:49638/Localization/Default.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

Helsinki
Moscow
Tokyo
Vienna
Warsaw
Washington D.C.

Figure 30-9

This is pretty much what you would expect. Now, however, change the previous example from Listing 30-12 so that the culture is set to Finnish, as shown in Listing 30-13.

Listing 30-13: Changing the culture to Finnish

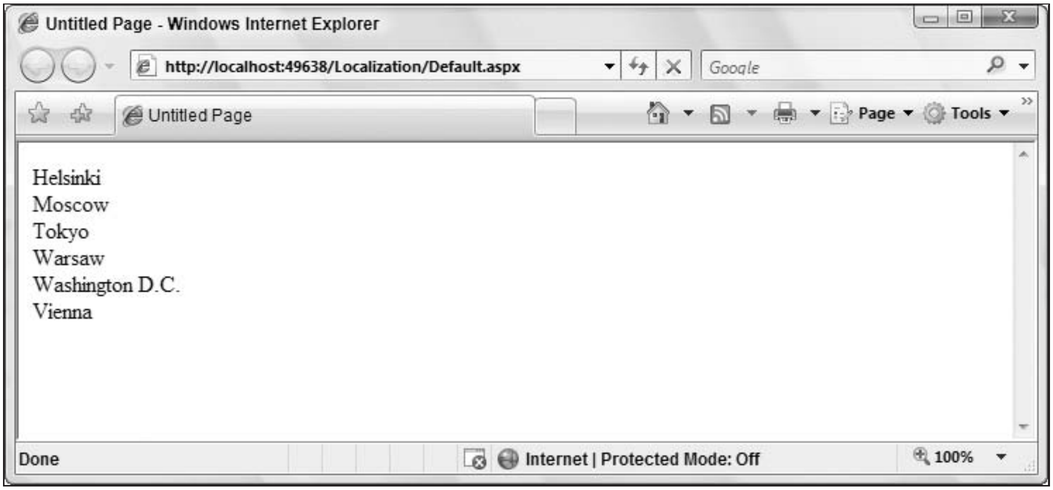
VB

```
System.Threading.Thread.CurrentThread.CurrentCulture = New CultureInfo("fi-FI")
```

C#

```
System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("fi-FI");
```

If you run the same bit of code under the Finnish culture setting, you get the results presented in Figure 30-10.



Untitled Page - Windows Internet Explorer
http://localhost:49638/Localization/Default.aspx
Google
Untitled Page
Page
Tools
Done
Internet | Protected Mode: Off
100%

Helsinki
Moscow
Tokyo
Warsaw
Washington D.C.
Vienna

Figure 30-10

If you examine the difference between the Finnish culture sorting done in Figure 30-10 and the U.S. English culture sorting done in Figure 30-9, you see that the city of Vienna is in a different place in the Finnish version. This is because, in the Finnish language, there is no difference between the letter V and the letter W. Because no difference exists, if you are sorting using the Finnish culture setting, then `Vi` comes after `Wa` and thus Vienna comes last in the list of strings in the sorting operation.

ASP.NET 3.5 Resource Files

When you work with ASP.NET 3.5, all resources are handled by a resource file. A resource file is an XML-based file that has a `.resx` extension. You can have Visual Studio 2008 help you construct this file. Resource files provide a set of items that are utilized by a specified culture. In your ASP.NET 3.5 applications, you store resource files as either local resources or global resources. The following sections look at how to use each type of resource.

Making Use of Local Resources

You would be surprised how easily you can build an ASP.NET page so that it can be *localized* into other languages. Really, the only thing you need to do is build the ASP.NET page as you normally would and then use some built-in capabilities from Visual Studio 2008 to convert the page to a format that allows you to plug in other languages easily.

To see this in action, build a simple ASP.NET page as presented in Listing 30-14.

Listing 30-14: Building the basic ASP.NET page to localize

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        Label2.Text = TextBox1.Text
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Sample Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server"
         Text="What is your name?"></asp:Label><br />
        <br />
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" Text="Submit Name" /><br />
        <br />
        <asp:Label ID="Label2" runat="server"></asp:Label>
    </div>
    </form>
</body>
</html>
```

As you can see, there is not much to this page. It is composed of a couple of Label controls, as well as TextBox and Button controls. The end user enters her name into the text box, and then the `Label2` server control is populated with the inputted name and a simple greeting.

The next step is what makes Visual Studio so great. To change the construction of this page so that it can be localized easily from resource files, open the page in Visual Studio and select Tools ➪ Generate Local Resource from the Visual Studio menu. Note that you can select this tool only when you are in the Design view of your page. It will not work in the split view or the code view of the page.

Selecting the Generate Local Resource from the Tool menu option causes Visual Studio to create an App_LocalResouces folder in your project if you already do not have one. A `.resx` file based upon this ASP.NET page is then placed in the folder. For instance, if you are working with the `Default.aspx` page, the resource file is named `Default.aspx.resx`. These changes are shown in Figure 30-11.

If you right-click on the `.resx` file and select View Code, notice that the `.resx` file is nothing more than an XML file with an associated schema at the beginning of the document. The resource file that is generated for you takes every possible property of every translatable control on the page and gives each item a key value that can be referenced in your ASP.NET page. If you look at the code of the page, notice that all the text values that you placed in the page have been left in the page, but they have also been placed inside the resource file. You can see how Visual Studio changed the code of the `Default.aspx` page in Listing 30-15.

Listing 30-15: Looking at how Visual Studio altered the page code

```
<%@ Page Language="VB" Culture="auto" meta:resourcekey="PageResource1"
    UICulture="auto" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        Label2.Text = TextBox1.Text
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Sample Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Text="What is your name?"
         meta:resourcekey="Label1Resource1"></asp:Label><br />
        <br />
        <asp:TextBox ID="TextBox1" runat="server"
         meta:resourcekey="TextBox1Resource1"></asp:TextBox>&nbsp;
        <asp:Button ID="Button1"
         runat="server" Text="Submit Name"
         meta:resourcekey="Button1Resource1" /><br />
        <br />
```

```
            <asp:Label ID="Label2" runat="server"
             meta:resourcekey="Label2Resource1"></asp:Label>
        </div>
        </form>
    </body>
    </html>
```

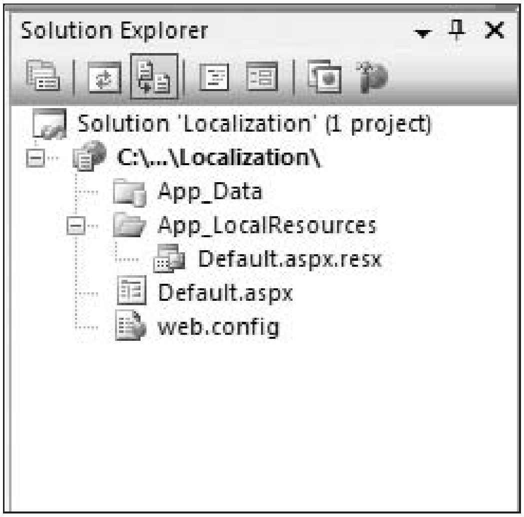


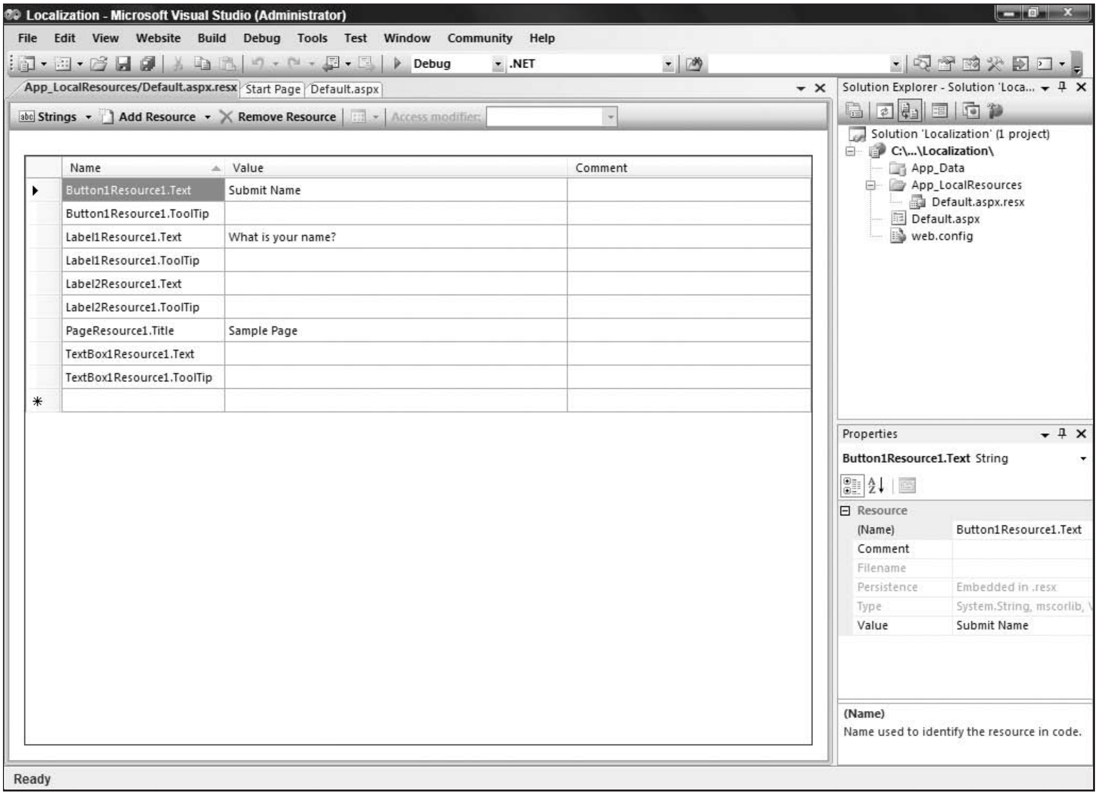
Figure 30-11

From this bit of code, you can see that the `Culture` and `UICulture` attributes have been added to the `@Page` directive with a value of `auto`, thus enabling this application to be localized. Also, the attribute `meta:resourcekey` has been added to each of the controls along with an associated value. This is the key from the `.resx` file that was created on your behalf. Double-clicking on the `Default.aspx.resx` file opens the resource file in the Resource Editor, which you will find is built into Visual Studio. This new editor is presented in Figure 30-12.

In the figure, note that a couple of properties from each of the server controls have been defined in the resource file. For instance, the Button server control has its `Text` and `ToolTip` properties exposed in this resource file, and the Visual Studio localization tool has pulled the default `Text` property value from the control based on what you placed there. Looking more closely at the Button server control constructions in this file, you can see that both the `Text` and `ToolTip` properties have a defining `Button1Resource1` value preceding the property name. This key is used in the Button server control you saw earlier.

```
<asp:Button ID="Button1"
 runat="server" Text="Submit Name"
 meta:resourcekey="Button1Resource1" />
```

You can see that a `meta:resourcekey` attribute has been added and, in this case, it references `Button1Resource1`. All the properties using this key in the resource file (for example, the `Text` and `ToolTip` properties) are applied to this Button server control at runtime.


Localization - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Tools Test Window Community Help
Debug
.NET
App_LocalResources/Default.aspx.resx
Start Page
Default.aspx
Strings
Add Resource
Remove Resource
Access modifier:
Solution Explorer - Solution 'Loca...
Solution 'Localization' (1 project)
C:\...\Localization\
App_Data
App_LocalResources
Default.aspx.resx
Default.aspx
web.config
Properties
Ready

Name	Value	Comment
Button1Resource1.Text	Submit Name	
Button1Resource1.ToolTip		
Label1Resource1.Text	What is your name?	
Label1Resource1.ToolTip		
Label2Resource1.Text		
Label2Resource1.ToolTip		
PageResource1.Title	Sample Page	
TextBox1Resource1.Text		
TextBox1Resource1.ToolTip		

Button1Resource1.Text String

Resource	
(Name)	Button1Resource1.Text
Comment	
Filename	
Persistence	Embedded in .resx
Type	System.String, mscorlib, \
Value	Submit Name

(Name)
Name used to identify the resource in code.

Figure 30-12

Adding Another Language Resource File

Now that the `Default.aspx.resx` file is in place, this is a file for an invariant culture. No culture is assigned to this resource file. If no culture can be determined, this resource file is then utilized. To add another resource file for the `Default.aspx` page that handles another language altogether, you copy and paste the `Default.aspx.resx` file into the same App_LocalResources folder and rename the newly copied file. If you use `Default.aspx.fi-FI.resx`, you give the following keys the following values to make a Finnish language resource file.

```
Button1Resource1.Text   Lähetä Nimi
Label1Resource1.Text    Mikä sinun nimi on?
PageResource1.Title     Näytesivu
```

You want to create a custom resource in both resource files using the key `Label2Answer`. The `Default.aspx.resx` file should have the following new key:

```
Label2Answer    Hello
```

Now you can add the key `Label2Answer` to the `Default.aspx.fi-FI.resx` file as shown here:

```
Label2Answer    Hei
```

You now have resources for specific controls and a resource that you can access later programmatically.

Finalizing the Building of the Default.aspx Page

Finalizing the `Default.aspx` page, you want to add a `Button1_Click` event so that when the end user enters a name into the text box and clicks the Submit button, the `Label2` server control provides a greeting to him or her that is pulled from the local resource files. When all is said and done, you should have a `Default.aspx` page that resembles the one in Listing 30-16.

Listing 30-16: The final Default.aspx page

VB

```
<%@ Page Language="VB" Culture="auto" meta:resourcekey="PageResource1"
    UICulture="auto" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)

       Label2.Text = GetLocalResourceObject("Label2Answer").ToString() & _
          " " & TextBox1.Text
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Sample Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Text="What is your name?"
         meta:resourcekey="Label1Resource1"></asp:Label><br />
        <br />
        <asp:TextBox ID="TextBox1" runat="server"
         meta:resourcekey="TextBox1Resource1"></asp:TextBox>&nbsp;
        <asp:Button ID="Button1"
         runat="server" Text="Submit Name"
         meta:resourcekey="Button1Resource1" OnClick="Button1_Click" /><br />
        <br />
        <asp:Label ID="Label2" runat="server"
         meta:resourcekey="Label2Resource1"></asp:Label>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" Culture="auto" meta:resourcekey="PageResource1"
    UICulture="auto" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
```

Continued

```
        {
            Label2.Text = GetLocalResourceObject("Label2Answer").ToString() + " " +
               TextBox1.Text;
        }
    </script>
```

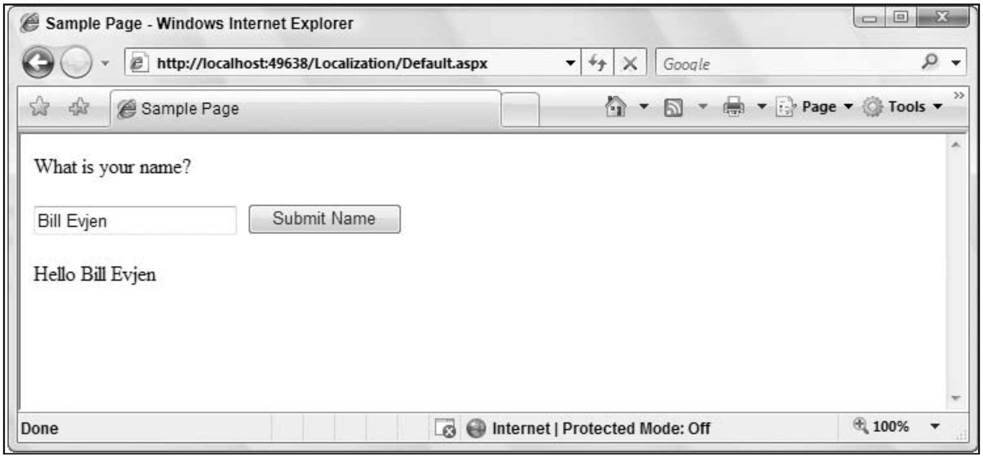
In addition to pulling local resources using the `meta:resourcekey` attribute in the server controls on the page to get at the exposed attributes, you can also get at any property value contained in the local resource file by using the `GetLocalResourceObject`. When using `GetLocalResourceObject`, you simply use the name of the key as a parameter as shown here:

```
GetLocalResourceObject("Label2Answer")
```

You could just as easily get at any of the controls property values from the resource file programmatically using the same construct:

```
GetLocalResourceObject("Button1Resource1.Text")
```

With the code from Listing 30-16 in place and the resource files completed, you can run the page, entering a name in the text box and then clicking the button to get a response, as illustrated in Figure 30-13.


Sample Page - Windows Internet Explorer
http://localhost:49638/Localization/Default.aspx
Google
Sample Page
Page
Tools

What is your name?

Bill Evjen

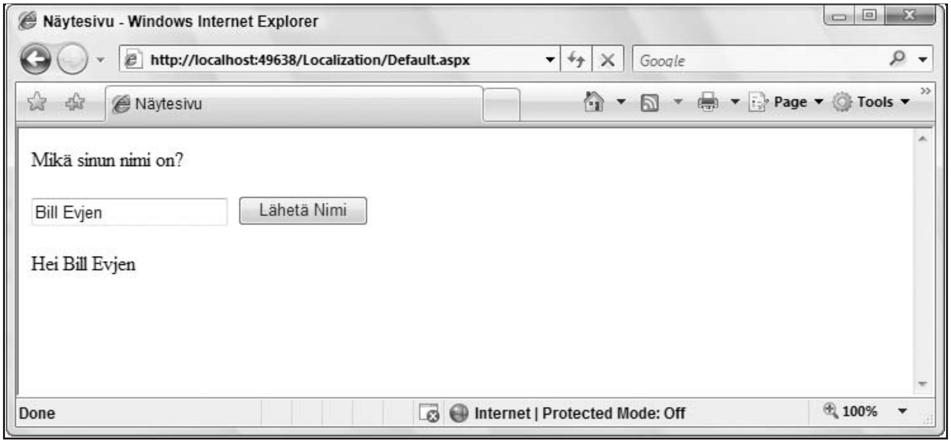
Hello Bill Evjen

Done
Internet | Protected Mode: Off
100%

Figure 30-13

What happened behind the scenes that caused this page to be constructed in this manner? First, only two resource files, `Default.aspx.resx` and `Default.aspx.fi-FI.resx`, are available. The `Default.aspx.resx` resource file is the invariant culture resource file, whereas the `Default.aspx.fi-FI.resx` resource file is for a specific culture (fi-FI). Because I requested the `Default.aspx` page and my browser is set to en-US as my preferred culture, ASP.NET found the local resources for the `Default.aspx` page. From there, ASP.NET made a check for an en-US–specific version of the `Default.aspx` page. Because there is not a specific page for the en-US culture, ASP.NET made a check for an EN (neutral culture) specific page. Not finding a page for the EN neutral culture, ASP.NET was then forced to use the invariant culture resource file of `Default.aspx.resx`, producing the page presented in Figure 30-13.

Now, if you set your IE language preference as fi-FI and rerun the `Default.aspx` page, you see a Finnish version of the page, as illustrated in Figure 30-14.


Näytesivu - Windows Internet Explorer
http://localhost:49638/Localization/Default.aspx
Google
Näytesivu
Page
Tools
Lähetä Nimi
Done
Internet | Protected Mode: Off
100%

Mikä sinun nimi on?

Bill Evjen

Hei Bill Evjen

Figure 30-14

In this case, having set my IE language preference to fi-FI, I am presented with this culture's page instead of the invariant culture page that was presented earlier. ASP.NET found this specific culture through use of the `Default.aspx.fi-FI.resx` resource file.

You can see that all the control properties that were translated and placed within the resource file are utilized automatically by ASP.NET including the page title presented in the title bar of IE.

Neutral Cultures Are Generally More Preferred

When you are working with the resource files from this example, note that one of the resources is for a *specific culture*. The `Default.aspx.fi-FI.resx` file is for a specific culture — the Finnish language as spoken in Finland. Another option would be to make this file work not for a specific culture, but instead for a neutral culture. To accomplish this task, you simply name the file `Default.aspx.FI.resx` instead. In this example, it really does not make that much difference because no other countries speak Finnish. It would make sense for languages such as German, Spanish, or French. These languages are spoken in multiple countries. For instance, if you are going to have a Spanish version of the `Default.aspx` page, you could definitely build it for a specific culture, such as `Default.aspx.es-MX.resx`. This construction is for the Spanish language as spoken in Mexico. With this in place, if someone requests the `Default.aspx` page with the language setting of es-MX, that user is provided with the contents of this resource file. However, what if the requestor has a setting of es-ESHe will not get the `Default.aspx.es-MX.resx` resource file, but instead gets the invariant culture resource file of `Default.aspx.resx`. If you are going to make only a single translation into German, Spanish, or another language for your site or any of your pages, you want to construct the resource files to be for neutral cultures rather than for specific cultures.

If you have the resource file `Default.aspx.ES.resx`, then it won't matter if the end user's preferred setting is set to es-MX, es-ES, or even es-AR — the user gets the appropriate ES neutral culture version of the page.

Making Use of Global Resources

Besides using only local resources that specifically deal with a particular page in your ASP.NET application, you also have the option of creating *global* resources that can be used across multiple pages. To create a resource file that can be utilized across the entire application, right-click on the solution in the Solution Explorer of Visual Studio and select Add New Item. From the Add New Item dialog, select Resource file.

Selecting this option provides you with a `Resource.resx` file. Visual Studio places this file in a new folder called App_GlobalResources. Again, this first file is the invariant culture resource file. Add a single string resource giving it the key of `PrivacyStatement` and a value of some kind (a long string).

After you have the invariant culture resource file completed, the next step is to add another resource file, but this time name it `Resource.fi-FI.resx`. Again, for this resource file, give a string key of `PrivacyStatement` and a different value altogether from the one you used in the other resource file.

The idea of a global resource file is that you have access to these resources across your entire application. You can gain access to the values that you place in these files in several ways. One way is to work the value directly into any of your server control declarations. For instance, you can place this privacy statement in a Label server control as presented in Listing 30-17.

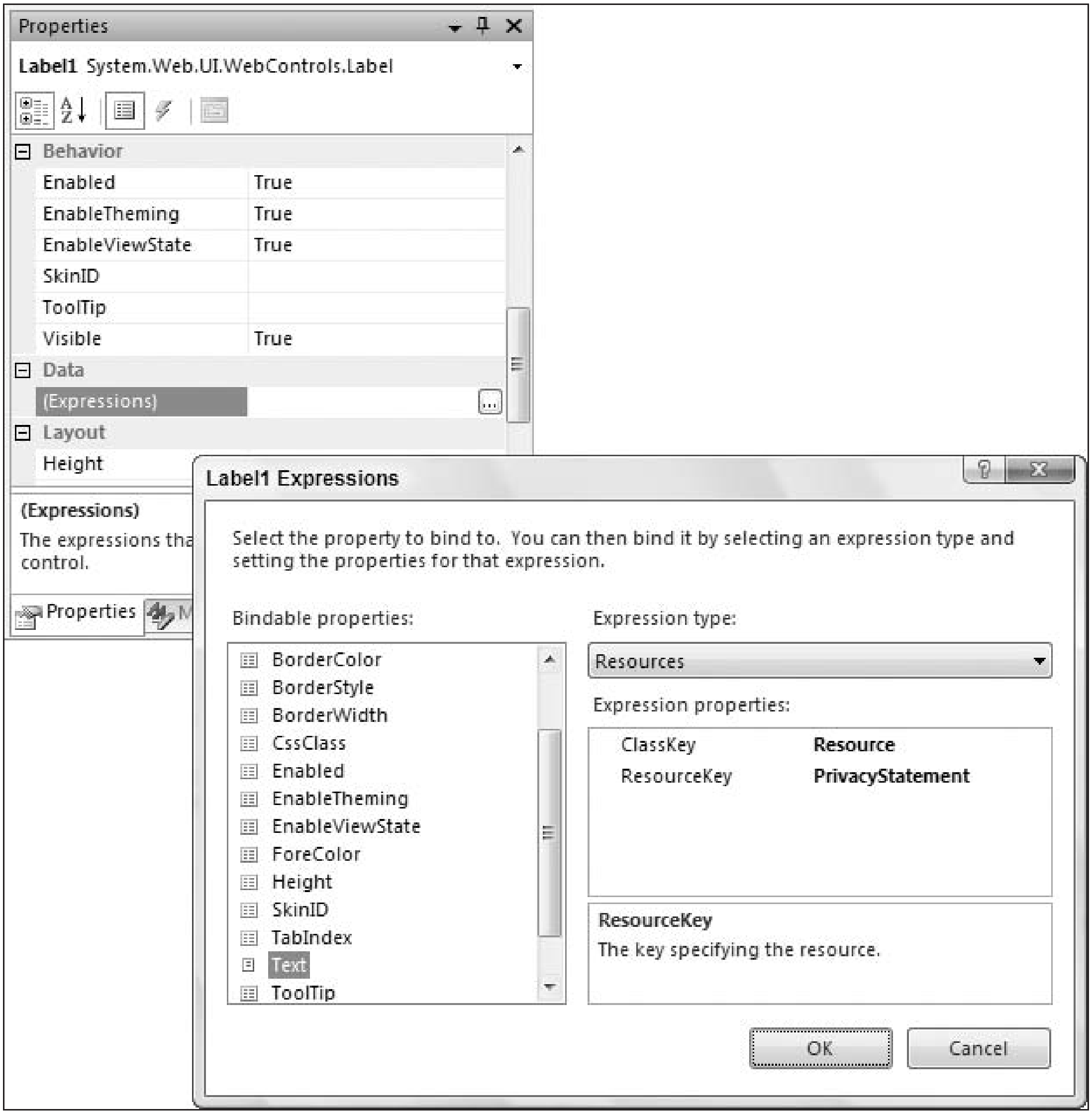
Listing 30-17: Using a global resource directly in a server control

```
<asp:Label ID="Label1" runat="server"
 Text='<%$ Resources: Resource, PrivacyStatement %>'></asp:Label>
```

With this construction in place, you can now grab the appropriate value of the `PrivacyStatement` global resource depending upon the language preference of the end user requesting the page. To make this work, you use the keyword Resources followed by a colon. Next, you specify the name of the resource file class. In this case, the name of the resource file is `Resource` because this statement goes to the `Resource.resx` and `Resource.fi-FI.resx` files in order to find what it needs. After specifying the particular resource file to use, the next item in the statement is the key — in this case, `PrivacyStatement`.

Another way of achieving the same result is to use some built-in dialogs within Visual Studio. To accomplish this task, highlight the server control you want in Visual Studio from Design view so that the control appears within the Properties window. For my example, I highlighted a Label server control. From the Properties window, you click the button within the Expressions property. This launches the Expressions dialog and enables you to bind the `PrivacyStatement` value to the `Text` property of the control. This is illustrated in Figure 30-15.

To make this work, highlight the Text property in the Bindable properties list. You then select an expression type from a drop-down list on the right-hand side of the dialog. Your options include AppSettings, ConnectionStrings, and Resources. Select the Resources option and you are then asked for the `ClassKey` and `ResourceKey` property values. The `ClassKey` is the name of the file that should be utilized. In this example, the name of the file is `Resource.resx`. Therefore, use the `Resource` keyword as a value. You are provided with a drop-down list in the `ResourceKey` property section with all the keys available in this file. Because only a single key exists at this point, you find only the `PrivacyStatement` key in this list. Make this selection and click the OK button. The Label server control changes and now appears as it was presented earlier in Listing 30-17.


Properties
Label1 System.Web.UI.WebControls.Label
Behavior
Data
Layout
Properties

Enabled	True
EnableTheming	True
EnableViewState	True
SkinID	
ToolTip	
Visible	True
(Expressions)	
Height	

(Expressions)
The expressions tha
control.

Label1 Expressions

Select the property to bind to. You can then bind it by selecting an expression type and setting the properties for that expression.

Bindable properties:
BorderColor
BorderStyle
BorderWidth
CssClass
Enabled
EnableTheming
EnableViewState
ForeColor
Height
SkinID
TabIndex
Text
ToolTip
OK
Cancel

Expression type: Resources

Expression properties:

ClassKey	**Resource**
ResourceKey	**PrivacyStatement**

ResourceKey
The key specifying the resource.

Figure 30-15

One nice feature is that the resources provided via global resources are available in a strongly typed manner. For instance, you can programmatically get at a global resource value by using the construction presented in Listing 30-18.

Listing 30-18: Programmatically getting at global resources

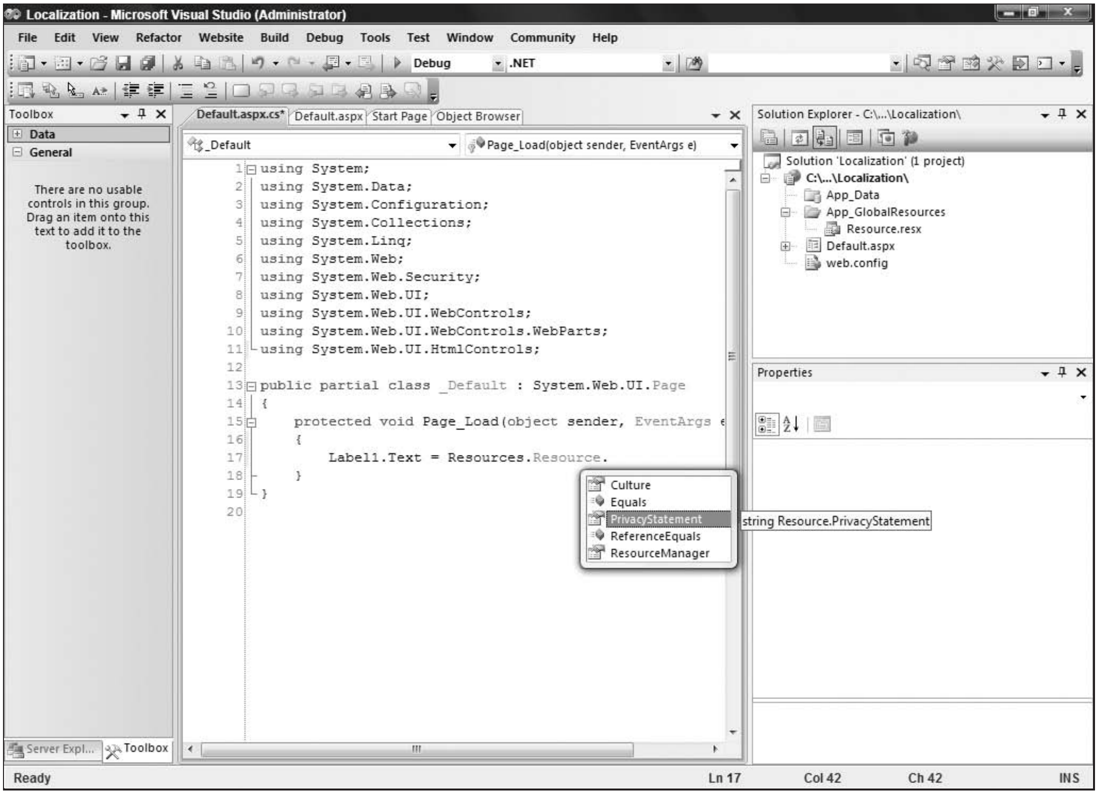
VB

```
Label1.Text = Resources.Resource.PrivacyStatement
```

C#

```
Label1.Text = Resources.Resource.PrivacyStatement;
```

In Figure 30-16, you can see that you have full intelliSense for these resource values.


Localization - Microsoft Visual Studio (Administrator)
File
Edit
View
Refactor
Website
Build
Debug
Tools
Test
Window
Community
Help
Debug
.NET
Toolbox
Data
General
Default.aspx.cs*
Default.aspx
Start Page
Object Browser
_Default
Page_Load(object sender, EventArgs e)
Solution Explorer - C:\...\Localization\
Solution 'Localization' (1 project)
C:\...\Localization\
App_Data
App_GlobalResources
Resource.resx
Default.aspx
web.config
Properties
Culture
Equals
PrivacyStatement
ReferenceEquals
ResourceManager
string Resource.PrivacyStatement
Server Expl...
Toolbox
Ready
Ln 17
Col 42
Ch 42
INS

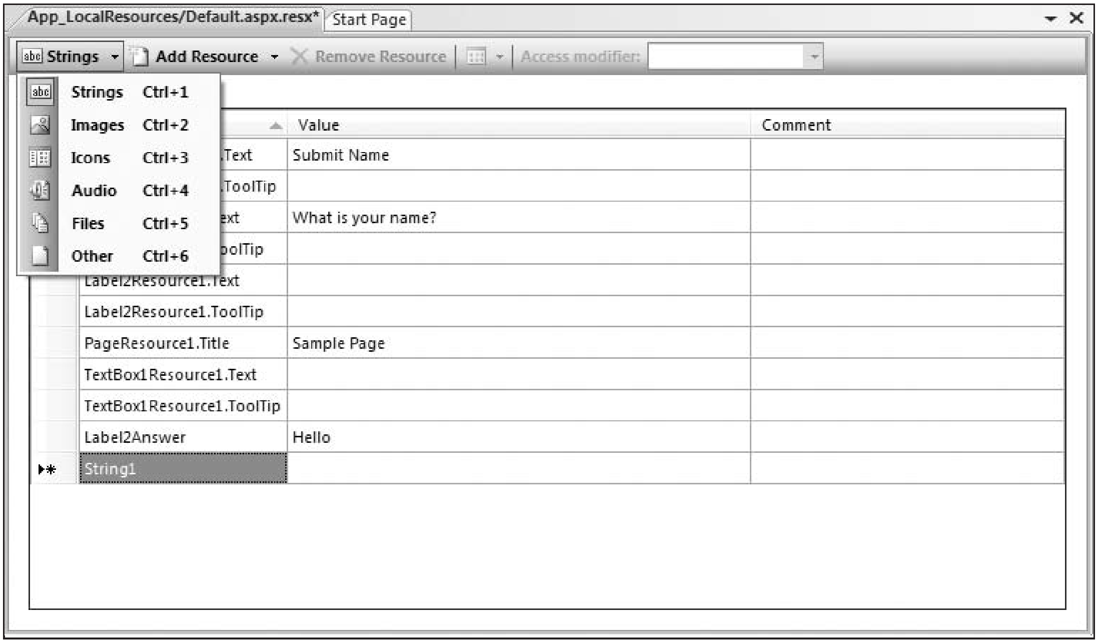
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e
    {
        Label1.Text = Resources.Resource.
    }
}
```

Figure 30-16

Looking at the Resource Editor

Visual Studio 2008 provides an editor for working with resource files. You have already seen some of the views available from this editor. Resources are categorized visually by the data type of the resource. So far, this chapter has dealt only with the handling of strings, but other categories exist (as well as images, icons, audio files, miscellaneous files, and other items). These options are illustrated in Figure 30-17.


App_LocalResources/Default.aspx.resx*
Start Page
Strings
Add Resource
Remove Resource
Access modifier:
Strings Ctrl+1
Images Ctrl+2
Icons Ctrl+3
Audio Ctrl+4
Files Ctrl+5
Other Ctrl+6

	Value	Comment
.Text	Submit Name	
.ToolTip		
ext	What is your name?	
olTip		
Label2Resource1.Text		
Label2Resource1.ToolTip		
PageResource1.Title	Sample Page	
TextBox1Resource1.Text		
TextBox1Resource1.ToolTip		
Label2Answer	Hello	
String1		

Figure 30-17

Summary

We hope you see the value in localizing your ASP.NET applications so that they can handle multiple cultures. This chapter looked at some of the issues you face when localizing your applications and some of the built-in tools provided via both Visual Studio and the .NET Framework to make this process easier for you.

It really is almost as easy as taking ASP.NET pages that have been already created and running the appropriate Visual Studio tool over the files to rebuild the pages to handle translations.

31

Configuration

Those of you who remember the ‘‘Classic’’ ASP days know that ASP’s configuration information was stored in a binary repository called the Internet Information Services (IIS) metabase. To configure a classic ASP application, you had to modify the metabase, either through script or, more commonly, through the IIS Microsoft Management Console (MMC) snap-in.

Unlike classic ASP, all the available versions of ASP.NET do not require extensive use of the IIS metabase. Instead, ASP.NET uses an XML file-based configuration system that is much more flexible, accessible, and easier to use. When building ASP.NET, the ASP.NET team wanted to improve the manageability of the product. Although the release of ASP.NET 1.0 was a huge leap forward in Web application development, it really targeted the developer. What was missing was the focus on the administrator — the person who takes care of Web applications after they are built and deployed. ASP.NET today makes it quite easy for you to configure an ASP.NET application by working either directly with the various configuration files or by using GUI tools that, in turn, interact with configuration files. Before examining the various GUI-based tools in detail in Chapter 34, you first take an in-depth look at how to work directly with the XML configuration files to change the behavior of your ASP.NET applications.

This chapter covers the following:

- ❑ Introduction to the ASP.NET configuration file
- ❑ An overview of the ASP.NET configuration settings
- ❑ How to encrypt portions of your configuration files
- ❑ An examination of the ASP.NET 3.5 configuration APIs
- ❑ How to store and retrieve sensitive information

The journey into these new configuration enhancements starts with an overview of configuration in ASP.NET.

Configuration Overview

ASP.NET configuration is stored in two primary XML-based files in a hierarchal fashion. XML is used to describe the properties and behaviors of various aspects of ASP.NET applications.

The ASP.NET configuration system supports two kinds of configuration files:

- Server or machine-wide configuration files such as the `machine.config` file
- Application configuration files such as the `web.config` file

Because the configuration files are based upon XML, the elements that describe the configuration are, therefore, case-sensitive. Moreover, the ASP.NET configuration system follows camel-casing naming conventions. If you look at the session state configuration example shown in Listing 31-1, for example, you can see that the XML element that deals with session state is presented as `<sessionState>`.

Listing 31-1: Session state configuration

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
<system.web>
    <sessionState
       mode="InProc"
       stateConnectionString="tcpip=127.0.0.1:42424"
       stateNetworkTimeout="10"
       sqlConnectionString="data source=127.0.0.1; user id=sa; password=P@55worD"
       cookieless="false"
       timeout="20" />
  </system.web>
</configuration>
```

The benefits of having an XML configuration file instead of a binary metabase include the following:

- The configuration information is human-readable and can be modified using a plain text editor such as Notepad, although it is recommended to use Visual Studio 2008 or another XML-aware editor. Unlike a binary metabase, the XML-based configuration file can be easily copied from one server to another, as with any simple file. This feature is extremely helpful when working in a Web farm scenario.
- When some settings are changed in the configuration file, ASP.NET automatically detects the changes and applies them to the running ASP.NET application. ASP.NET accomplishes this by creating a new instance of the ASP.NET application and directing end users to this new application.
- The configuration changes are applied to the ASP.NET application without the need for the administrator to stop and start the Web server. Changes are completely transparent to the end user.
- The ASP.NET configuration system is extensible.
- Application-specific information can be stored and retrieved very easily.
- The sensitive information stored in the ASP.NET configuration system can optionally be encrypted to keep it from prying eyes.

Server Configuration Files

Every ASP.NET server installation includes a series of configuration files, such as the `machine.config` file. This file is installed as a part of the default .NET Framework installation. You can find `machine.config` and the other server-specific configuration files in `C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG`. They represent the default settings used by all ASP.NET Web applications installed on the server.

Some of the server-wide configuration files include the following:

- ❑ `machine.config`
- ❑ `machine.config.comments`
- ❑ `machine.config.default`
- ❑ `web.config`
- ❑ `web.config.comments`
- ❑ `web.config.default`
- ❑ `web_hightrust.config`
- ❑ `web_hightrust.config.default`
- ❑ `web_lowtrust.config`
- ❑ `web_lowtrust.config.default`
- ❑ `web_mediumtrust.config`
- ❑ `web_mediumtrust.config.default`
- ❑ `web_minimaltrust.config`
- ❑ `web_minimaltrust.config.default`

The system-wide configuration file, `machine.config`, is used to configure common .NET Framework settings for all applications on the machine. As a rule, it is not a good idea to edit or manipulate the `machine.config` file unless you know what you are doing. Changes to this file can affect all applications on your computer (Windows, Web, and so on).

> *Because the .NET Framework supports side-by-side execution mode, you might find more than one installation of the machine.config file if you have multiple versions of the .NET Framework installed on the server. If you have .NET Framework versions 1.0, 1.1, and 2.0 running on the server, for example, each .NET Framework installation has its own machine.config file. This means that you will find three machine.config file installations on that particular server. It is interesting to note that the .NET Framework 3.5 is really just a bolt-on to the .NET Framework 2.0. (It includes extra DLLs, which are sometimes referred to as extensions.) Thus, the .NET Framework 3.5 uses the same machine.config file as the .NET Framework 2.0.*

In addition to the `machine.config` file, the .NET Framework installer also installs two more files called `machine.config.default` and `machine.config.comments`. The `machine.config.default` file acts as a backup for the `machine.config` file. If you want to revert to the factory setting for `machine.config`, simply copy the settings from the `machine.config.default` to the `machine.config` file.

The `machine.config.comments` file contains a description for each configuration section and explicit settings for the most commonly used values. `machine.config.default` and `machine.config.comment` files are not used by the .NET Framework runtime; they're installed in case you want to revert back to default factory settings and default values.

You will also find a root-level `web.config` file in place within the same CONFIG folder as the `machine.config`. When making changes to settings on a server-wide basis, you should always attempt to make these changes in the root `web.config` file rather than in the `machine.config` file. You will find that files like the `machine.config.comments` and the `machine.config.default` files also exist for the `web.config` (`web.config.comments` and `web.config.default`).

By default, your ASP.NET Web applications run under a *full trust* setting. You can see this by looking at the `<securityPolicy>` and `<trust>` sections in the root-level `web.config` file. This section is presented in Listing 31-2.

Listing 31-2: The root web.config showing the trust level

```
<configuration>

    <location allowOverride="true">
        <system.web>
            <securityPolicy>
                <trustLevel name="Full" policyFile="internal" />
                <trustLevel name="High" policyFile="web_hightrust.config" />
                <trustLevel name="Medium" policyFile="web_mediumtrust.config" />
                <trustLevel name="Low"  policyFile="web_lowtrust.config" />
                <trustLevel name="Minimal" policyFile="web_minimaltrust.config" />
            </securityPolicy>
            <trust level="Full" originUrl="" />
        </system.web>
    </location>

</configuration>
```

The other policy files are defined at specific trust levels. These levels determine the code-access security (CAS) allowed for ASP.NET. To change the trust level in which ASP.NET applications can run on the server, you simply change the `<trust>` element within the document or within your application's instance of the `web.config` file. For example, you can change to a medium trust level using the code shown in Listing 31-3.

Listing 31-3: Changing the trust level to medium trust

```
<configuration>

    <location allowOverride="false">
        <system.web>
            <securityPolicy>
                <trustLevel name="Full" policyFile="internal" />
                <trustLevel name="High" policyFile="web_hightrust.config" />
                <trustLevel name="Medium" policyFile="web_mediumtrust.config" />
                <trustLevel name="Low" policyFile="web_lowtrust.config" />
                <trustLevel name="Minimal" policyFile="web_minimaltrust.config" />
            </securityPolicy>
            <trust level="Medium" originUrl="" />
        </system.web>
    </location>

</configuration>
```

In this case, not only does this code mandate use of the `web_mediumtrust.config` file, but also (by setting the `allowOverride` attribute to `false`) it forces this trust level upon every ASP.NET application on the server. Individual application instances are unable to change this setting by overriding it in their local `web.config` files because this setting is in the root-level `web.config` file.

If you look through the various trust level configuration files (such as the `web_mediumtrust.config` file), notice that they define what kinds of actions you can perform through your code operations. For instance, the `web_hightrust.config` file allows for open FileIO access to any point on the server as illustrated in Listing 31-4.

Listing 31-4: The web_hightrust.config file's definition of FileIO CAS

```
<IPermission
   class="FileIOPermission"
   version="1"
   Unrestricted="true"
/>
```

If, however, you look at the medium trust `web.config` file (`web_mediumtrust.config`), you see that this configuration file restricts ASP.NET to *only* those FileIO operations within the application directory. This definition is presented in Listing 31-5.

Listing 31-5: FileIO restrictions in the web_mediumtrust.config file

```
<IPermission
   class="FileIOPermission"
   version="1"
   Read="\$AppDir\$"
   Write="\$AppDir\$"
   Append="\$AppDir\$"
   PathDiscovery="\$AppDir\$"
/>
```

It is always a good idea to see in which trust level you can run your ASP.NET applications and to change the `<trust>` section to enable the appropriate level of CAS.

Application Configuration File

Unlike the `machine.config` file, each and every ASP.NET application has its own copy of configuration settings stored in a file called `web.config`. If the Web application spans multiple subfolders, each subfolder can have its own `web.config` file that inherits or overrides the parent's file settings.

To update servers in your farm with these new settings, you simply copy this `web.config` file to the appropriate application directory. ASP.NET takes care of the rest — no server restarts and no local server access is required — and your application continues to function normally, except that it now uses the new settings applied in the configuration file.

How Configuration Settings Are Applied

When the ASP.NET runtime applies configuration settings for a given Web request, `machine.config` (as well as any of the `web.config` files configuration information) is merged into a single unit, and that information is then applied to the given application. Configuration settings are inherited from any parent `web.config` file or `machine.config`, which is the root configuration file or the ultimate parent. An example of this is presented in Figure 31-1.

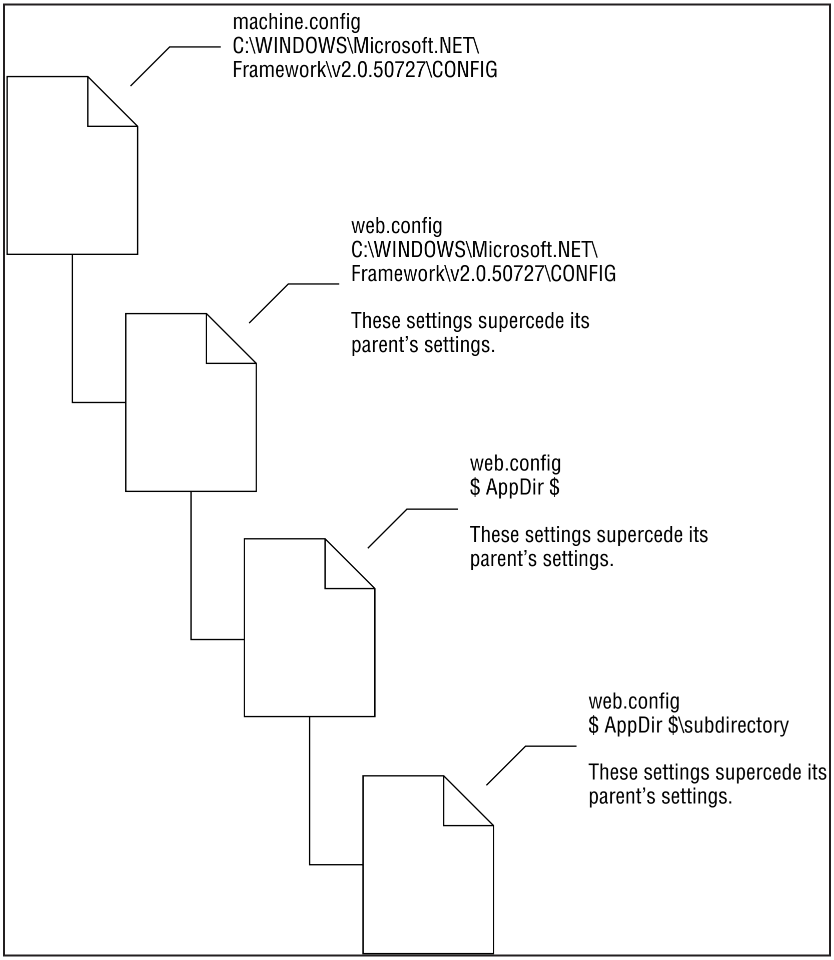

machine.config
C:\WINDOWS\Microsoft.NET\
Framework\v2.0.50727\CONFIG
web.config
C:\WINDOWS\Microsoft.NET\
Framework\v2.0.50727\CONFIG
These settings supercede its parent's settings.
web.config
$ AppDir $
These settings supercede its parent's settings.
web.config
$ AppDir $\subdirectory
These settings supercede its parent's settings.

Figure 31-1

The configuration for each Web application is unique; however, settings are inherited from the parent. For example, if the `web.config` file in the root of your Web site defines a session timeout of 10 minutes, then that particular setting overrides the default ASP.NET setting inherited from the `machine.config` or the root `web.config` file. The `web.config` files in the subdirectories or subfolders can override these settings or inherit the settings (such as the 10-minute session timeout).

The configuration settings for virtual directories are independent of the physical directory structure. Unless the manner in which the virtual directories are organized is exclusively specified, configuration problems can result.

Note that these inheritance/override rules can be blocked in most cases by using the `allowOverride = "false"` mechanism shown in Listing 31-3.

Detecting Configuration File Changes

ASP.NET automatically detects when configuration files, such as `machine.config` or `web.config`, are changed. This logic is implemented based on listening for file-change notification events provided by the operating system.

When an ASP.NET application is started, the configuration settings are read and stored in the ASP.NET cache. A file dependency is then placed on the entry within the cache in the `machine.config` and/or `web.config` configuration file. When the configuration file update is detected in the `machine.config`, ASP.NET creates a new application domain to service new requests. The old application domain is destroyed as soon as it completes servicing all its outstanding requests.

Configuration File Format

The main difference between `machine.config` and `web.config` is the filename. Other than that, their schemas are the same. Configuration files are divided into multiple groups. The root-level XML element in a configuration file is named `<configuration>`. This pseudo `web.config` file has a section to control ASP.NET, as shown in Listing 31-6.

Listing 31-6: A pseudo web.config file

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <configSections>
    <section name="[sectionSettings]" type="[Class]"/>
    <sectionGroup name="[sectionGroup]">
      <section name="[sectionSettings]" type="[Class]"/>
    </sectionGroup>
  </configSections>
</configuration>
```

Values within brackets [ ] have unique values within the real configuration file.

The root element in the XML configuration file is always `<configuration>`. Each of the section handlers and settings are optionally wrapped in a `<sectionGroup>`. A `<sectionGroup>` provides an organizational function within the configuration file. It allows you to organize configuration into unique groups — for instance, the `<system.web>` section group is used to identify areas within the configuration file specific to ASP.NET.

Config Sections

The `<configSections>` section is the mechanism to group the configuration section handlers associated with each configuration section. When you want to create your own section handlers, you must declare them in the `<configSections>` section. The `<httpModules>` section has a configuration handler that is set to `System.Web.Caching.HttpModulesSection`, and the `<sessionState>` section has a configuration handler that is set to `System.Web.SessionState.SessionStateSection` classes, as shown in Listing 31-7.

Listing 31-7: HTTP Module configuration setting from the machine.config file

```
<configSections>
   <sectionGroup>
      <section name="httpModules"
       type="System.Web.Configuration.HttpModulesSection,
          System.Web, Version=2.0.0.0, Culture=neutral,
          PublicKeyToken=b03f5f7f11d50a3a"/>
   </sectionGroup>
</configSections>
```

Common Configuration Settings

The ASP.NET applications depend on a few common configuration settings. These settings are common to both the `web.config` and `machine.config` files. In this section, you look at some of these common configuration settings.

Connecting Strings

In ASP.NET 1.0 and 1.1, all the connection string information was stored in the `<appSettings>` section. However, ever since ASP.NET 2.0, a section called `<connectionStrings>` was included that stores all kinds of connection-string information. Even though storing connection strings in the `<appSettings>` element works fine, it poses the following challenges:

- ❑ When connection strings are stored in `appSettings` section, it is impossible for a data-aware control such as `SqlCacheDependency` or `MembershipProvider` to discover the information.
- ❑ Securing connection strings using cryptographic algorithms is a challenge.
- ❑ Last, but not least, this feature does not apply to ASP.NET only; rather, it applies to all the .NET application types including Windows Forms, Web Services, and so on.

Because the connection-string information is stored independently of the `appSettings` section, it can be retrieved using the strongly typed collection method `ConnectionStrings`. Listing 31-8 gives an example of how to store connecting strings.

Listing 31-8: Storing a connection string

```
<configuration>
  <connectionStrings>
    <add
        name = "ExampleConnection"
        connectionString = "server=401kServer;database=401kDB;
        uid=WebUser;pwd=P@$$worD9" />
  </connectionStrings>
</configuration>
```

Listing 31-9 shows how to retrieve the connection string (`ExampleConnection`) in your code.

Listing 31-9: Retrieving a connection string

VB

```
Public Sub Page_Load (sender As Object, e As EventArgs)
    ...
    Dim dbConnection as New _
      SqlConnection(ConfigurationManager.ConnectionStrings("ExampleConnection")
             .ConnectionString)
    ...
End Sub
```

C#

```
public void Page_Load (Object sender, EventArgs e)
{
    ...
    SqlConnection dbConnection = new
      SqlConnection(ConfigurationManager.ConnectionStrings["ExampleConnection"]
             .ConnectionString);
    ...
}
```

This type of construction has a lot of power. Instead of hard-coding your connection strings into each and every page within your ASP.NET application, you can store one instance of the connection string centrally (in the `web.config` file for instance). Now, if you have to make a change to this connection string, you can make this change in only *one* place rather than in multiple places.

Configuring Session State

Because Web-based applications utilize the stateless HTTP protocol, you must store the application-specific state or user-specific state where it can persist. The `Session` object is the common store where user-specific information is persisted. Session store is implemented as a `Hashtable` and stores data based on key/value pair combinations.

ASP.NET 1.0 and 1.1 had the capability to persist the session store data in `InProc`, `StateServer`, and `SqlServer`. ASP.NET 2.0 and 3.5 adds one more capability called `Custom`. The `Custom` setting gives the developer a lot more control regarding how the session state is persisted in a permanent store. For example, out of the box ASP.NET 3.5 does not support storing session data on non-Microsoft databases such as Oracle, DB2, or Sybase. If you want to store the session data in any of these databases or in a custom store such as an XML file, you can implement that by writing a custom provider class. (See the section ''Custom State Store'' later in this chapter and Chapter 22 to learn more about the new session state features in ASP.NET 3.5.)

You can configure the session information using the `<sessionState>` element as presented in Listing 31-10.

Listing 31-10: Configuring session state

```
<sessionState
 mode="StateServer"
 cookieless="false"
 timeout="20"
 stateConnectionString="tcpip=ExampleSessionStore:42424"
 stateNetworkTimeout="60"
 sqlConnectionString=""
/>
```

The following list describes each of the attributes for the `<sessionState>` element shown in the preceding code:

- `mode`: Specifies whether the session information should be persisted. The mode setting supports five options: `Off`, `InProc`, `StateServer`, `SQLServer`, and `Custom`. The default option is `InProc`.
- `cookieless`: Specifies whether HTTP cookieless Session key management is supported.
- `timeout`: Specifies the `Session` lifecycle time. The `timeout` value is a sliding value; at each request, the timeout period is reset to the current time plus the timeout value. For example, if the `timeout` value is 20 minutes and a request is received at 10:10 AM, the timeout occurs at 10:30 AM.
- `stateConnectionString`: When `mode` is set to `StateServer`, this setting is used to identify the TCP/IP address and port to communicate with the Windows Service providing state management.
- `stateNetworkTimeout`: Specifies the timeout value (in seconds) while attempting to store state in an out-of-process session store such as `StateServer`.
- `sqlConnectionString`: When `mode` is set to `SQLServer`, this setting is used to connect to the SQL Server database to store and retrieve session data.

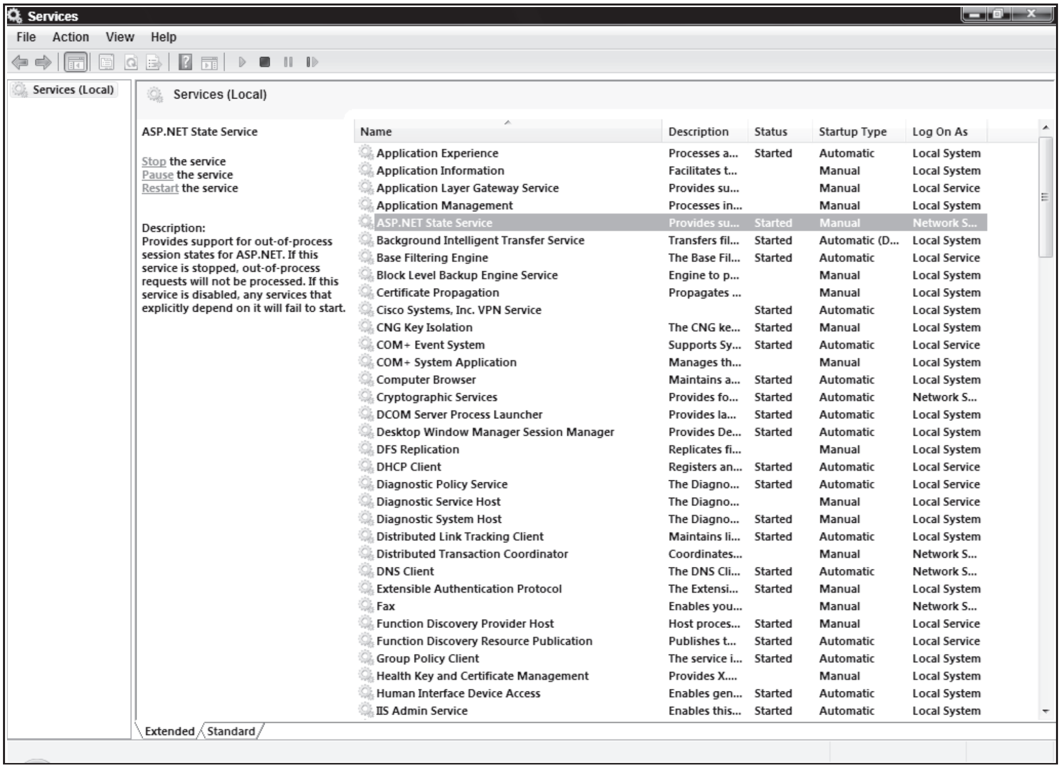
Web Farm Support

Multiple Web servers working as a group are called a Web Farm. If you would like to scale out your ASP.NET application into multiple servers inside a Web Farm, ASP.NET supports this kind of deployment out of the box. However, the session data needs to be persisted in an out-of-process session state such as `StateServer` or `SQLServer`.

State Server

Both `StateServer` and `SQLServer` support the out-of-process session state. However, the `StateServer` stores all the session information in a Windows Service, which stores the session data in memory. Using this option, if the server that hosts the session state service goes down in the Web farm, all the ASP.NET clients that are accessing the Web site fail; there is no way to recover the session data.

You can configure the session state service using the Services dialog available by choosing Start ➪ Settings ➪ Control Panel ➪ Administrative Tools ➪ Computer Management if you are using Windows XP, and Start ➪ Control Panel ➪ System and Maintenance ➪ Administrative Tools ➪ Services if you are using Windows Vista (as shown in Figure 31-2).



ASP.NET State Service

Stop the service
Pause the service
Restart the service

Description:
Provides support for out-of-process session states for ASP.NET. If this service is stopped, out-of-process requests will not be processed. If this service is disabled, any services that explicitly depend on it will fail to start.

Name	Description	Status	Startup Type	Log On As
Application Experience	Processes a...	Started	Automatic	Local System
Application Information	Facilitates t...		Manual	Local System
Application Layer Gateway Service	Provides su...		Manual	Local Service
Application Management	Processes in...		Manual	Local System
ASP.NET State Service	Provides su...	Started	Manual	Network S...
Background Intelligent Transfer Service	Transfers fil...	Started	Automatic (D...	Local System
Base Filtering Engine	The Base Fil...	Started	Automatic	Local Service
Block Level Backup Engine Service	Engine to p...		Manual	Local System
Certificate Propagation	Propagates ...		Manual	Local System
Cisco Systems, Inc. VPN Service		Started	Automatic	Local System
CNG Key Isolation	The CNG ke...	Started	Manual	Local System
COM+ Event System	Supports Sy...	Started	Automatic	Local Service
COM+ System Application	Manages th...		Manual	Local System
Computer Browser	Maintains a...	Started	Automatic	Local System
Cryptographic Services	Provides fo...	Started	Automatic	Network S...
DCOM Server Process Launcher	Provides la...	Started	Automatic	Local System
Desktop Window Manager Session Manager	Provides De...	Started	Automatic	Local System
DFS Replication	Replicates fi...		Manual	Local System
DHCP Client	Registers an...	Started	Automatic	Local Service
Diagnostic Policy Service	The Diagno...	Started	Automatic	Local Service
Diagnostic Service Host	The Diagno...		Manual	Local Service
Diagnostic System Host	The Diagno...	Started	Manual	Local System
Distributed Link Tracking Client	Maintains li...	Started	Automatic	Local System
Distributed Transaction Coordinator	Coordinates...		Manual	Network S...
DNS Client	The DNS Cli...	Started	Automatic	Network S...
Extensible Authentication Protocol	The Extensi...	Started	Manual	Local System
Fax	Enables you...		Manual	Network S...
Function Discovery Provider Host	Host proces...	Started	Manual	Local Service
Function Discovery Resource Publication	Publishes t...	Started	Automatic	Local Service
Group Policy Client	The service i...	Started	Automatic	Local System
Health Key and Certificate Management	Provides X....		Manual	Local System
Human Interface Device Access	Enables gen...	Started	Automatic	Local System
IIS Admin Service	Enables this...	Started	Automatic	Local System

Figure 31-2

Alternatively, you can start the session state service by using the command prompt and entering the `net start` command, like this:

```
C:\Windows\Microsoft.NET\Framework\v2.0.50727\> net start aspnet_state

The ASP.NET State Service service is starting.

The ASP.NET State Service service was started successfully.
```

All compatible versions of ASP.NET share a single state service instance, which is the service installed with the highest version of ASP.NET. For example, if you have installed ASP.NET 2.0 on a server where ASP.NET 1.0 and 1.1 are already running, the ASP.NET 2.0 installation replaces the ASP.NET 1.1's state server instance. The ASP.NET 2.0 service is guaranteed to work for all previous compatible versions of ASP.NET.

SQL Server

When you choose the `SQLServer` option, session data is stored in a Microsoft SQL Server database. Even if SQL Server goes down, the built-in SQL Server recovery features enable you to recover all the session data. Configuring ASP.NET to support SQL Server for session state is just as simple as configuring the Windows Service. The only difference is that you run a T-SQL script that ships with ASP.NET, `InstallSqlState.sql`. The T-SQL script that uninstalls ASP.NET SQL Server support, called `UninstallSqlState.sql`, is also included. The install and uninstall scripts are available in the Framework folder. An example of using the SQL Server option is presented in Listing 31-11.

Listing 31-11: Using the SQLServer option for session state

```
<configuration>
  <system.web>
    <sessionState
      mode="SQLServer"
      sqlConnectionString="data source=ExampleSessionServer;
      user id=ExampleWebUser;password=P@55worD"
      cookieless="false"
      timeout="20"
    />
  </system.web>
</configuration>
```

ASP.NET accesses the session data stored in SQL Server via stored procedures. By default, all the session data is stored in the Temp DB database. However, you can modify the stored procedures so they are stored in tables in a full-fledged database other than Temp DB.

> *Even though the SQL Server–based session state provides a scalable use of session state, it could become the single point of failure. This is because SQL Server session state uses the same SQL Server database for all applications in the same ASP.NET process. This problem has been fixed ever since ASP.NET 2.0, and you can configure different databases for each application. Now you can use the* `aspnet_regsql.exe` *utility to configure this. However, if you are looking for a solution for older .NET Frameworks, a fix is available at* `http://support.microsoft.com/default.aspx?scid = kb;EN-US;836680`.

Because the connection strings are stored in the strongly typed mode, the connection string information can be referenced in other parts of the configuration file. For example, when configuring session state to be stored in SQL Server, you can specify the connection string in the `connectionStrings` section, and then you can specify the name of the connection string in the `<sessionState>` element, as shown in Listing 31-12.

Listing 31-12: Configuring session state with a connection string

```
<configuration>

  <connectionStrings>
    <add name = "ExampleSqlSessionState"
         connectionString = "data source=ExampleSessionServer;
         user id=ExampleWebUser;password=P@55worD" />
  </connectionStrings>

  <system.web>
    <sessionState
      mode="SQLServer"
      sqlConnectionString="ExampleSqlSessionState"
```

```
        cookieless="false"
        timeout="20"
      />
    </system.web>

  </configuration>
```

Custom State Store

The session state in ASP.NET 3.5 is based on a pluggable architecture with different providers that inherit the `SessionStateStoreProviderBase` class. If you want to create your own custom provider or use a third-party provider, you must set the mode to `Custom`.

You specify the custom provider assembly that inherits the `SessionStateStoreProviderBase` class, as shown in Listing 31-13.

Listing 31-13: Working with your own session state provider

```
  <configuration>
     <system.web>

        <sessionState
            mode="Custom"
            customProvider="CustomStateProvider">
           <providers>
             <add name="CustomStateProvider"
              type="CustomStateProviderAssembly,
                CustomStateProviderNamespace.CustomStateProvider"/>
           </providers>
        </sessionState>

     </system.web>
  </configuration>
```

In the previous example, you have configured the session state mode as custom because you have specified the provider name as `CustomStateProvider`. From there, you add the provider element and include the type of the provider with namespace and class name.

You can read more about the provider model and custom providers in Chapters 12 and 13.

Compilation Configuration

ASP.NET supports the dynamic compilation of ASP.NET pages, Web services, HttpHandlers, ASP.NET application files (such as the `Global.asax` file), source files, and so on. These files are automatically compiled on demand when they are first required by an ASP.NET application.

Any changes to a dynamically compiled file causes all affected resources to become automatically invalidated and recompiled. This system enables developers to quickly develop applications with a minimum of process overhead because they can just press Save to immediately cause code changes to take effect within their applications.

The ASP.NET 1.0 and 1.1 features are extended in ASP.NET 2.0 and 3.5 to account for other file types, including class files. The ASP.NET compilation settings can be configured using the `<compilation>` section in `web.config` or `machine.config`. The ASP.NET engine compiles the page when necessary and saves the generated code in code cache. This cached code is used when executing the ASP.NET pages. Listing 31-14 shows the syntax for the `<compilation>` section.

Listing 31-14: The compilation section

```
<!-- compilation Attributes -->
<compilation
  tempDirectory="directory"
  debug="[true|false]"
  strict="[true|false]"
  explicit="[true|false]"
  batch="[true|false]"
  batchTimeout="timeout in seconds"
  maxBatchSize="max number of pages per batched compilation"
  maxBatchGeneratedFileSize="max combined size in KB"
  numRecompilesBeforeAppRestart="max number of recompilations "
  defaultLanguage="name of a language as specified in a <compiler/> element below"
   <compilers>
      <compiler language="language"
               extension="ext"
               type=".NET Type"
               warningLevel="number"
               compilerOptions="options"/>
   </compilers>
   <assemblies>
      <add assembly="assembly"/>
   </assemblies>
   <codeSubDirectories>
     <codeSubDirectory directoryName="sub-directory name"/>
   </codeSubDirectories>
   <buildproviders>
     <buildprovider
       extension="file extension"
       type="type reference"/>
   </buildproviders>
</compilation>
```

Now take a more detailed look at these `<compilation>` attributes:

- `batch`: Specifies whether the batch compilation is supported. The default value is `true`.
- `maxBatchSize`: Specifies the maximum number of pages/classes that can be compiled into a single batch. The default value is 1000.
- `maxBatchGeneratedFileSize`: Specifies the maximum output size of a batch assembly compilation. The default value is 1000 KB.
- `batchTimeout`: Specifies the amount of time (minutes) granted for batch compilation to occur. If this timeout elapses without compilation being completed, an exception is thrown. The default value is 15 minutes.
- `debug`: Specifies whether to compile production assemblies or debug assemblies. The default is `false`.

- `defaultLanguage`: Specifies the default programming language, such as `VB` or `C#`, to use in dynamic compilation files. Language names are defined using the `<compiler>` child element. The default value is `VB`.
- `explicit`: Specifies whether the Microsoft Visual Basic code compile option is explicit. The default is `true`.
- `numRecompilesBeforeAppRestart`: Specifies the number of dynamic recompiles of resources that can occur before the application restarts.
- `strict`: Specifies the setting of the Visual Basic strict compile option.
- `tempDirectory`: Specifies the directory to use for temporary file storage during compilation. By default, ASP.NET creates the temp file in the `[WinNT\Windows]\Microsoft.NET\Framework\[version]\Temporary ASP.NET Files` folder.
- `compilers`: The `<compilers>` section can contain multiple `<compiler>` subelements, which are used to create a new compiler definition:
    - The `language` attribute specifies the languages (separated by semicolons) used in dynamic compilation files. For example, `C#; VB`.
    - The `extension` attribute specifies the list of filename extensions (separated by semicolons) used for dynamic code. For example, `.cs; .vb`.
    - The `type` attribute specifies .NET type/class that extends the `CodeDomProvider` class used to compile all resources that use either the specified language or the file extension.
    - The `warningLevel` attribute specifies how the .NET compiler should treat compiler warnings as errors. Five levels of compiler warnings exist, numbered 0 through 4. When the compiler transcends the warning level set by this attribute, compilation fails. The meaning of each warning level is determined by the programming language and compiler you're using; consult the reference specification for your compiler to get more information about the warning levels associated with compiler operations and what events trigger compiler warnings.
    - The `compilerOptions` attribute enables you to include compiler's command-line switches while compiling the ASP.NET source.
- `assemblies`: Specifies assemblies that are used during the compilation process.
- `codeSubDirectories`: Specifies an ordered collection of subdirectories containing files compiled at runtime. Adding the `codeSubDirectories` section creates separate assemblies.
- `buildproviders`: Specifies a collection of build providers used to compile custom resource files.

Browser Capabilities

Identifying and using the browser's capabilities is essential for Web applications. The browser capabilities component was designed for the variety of desktop browsers, such as Microsoft's Internet Explorer, Netscape, Opera, and so on. The `<browserCaps>` element enables you to specify the configuration settings for the browser capabilities component. The `<browserCaps>` element can be declared at the machine, site, application, and subdirectory level.

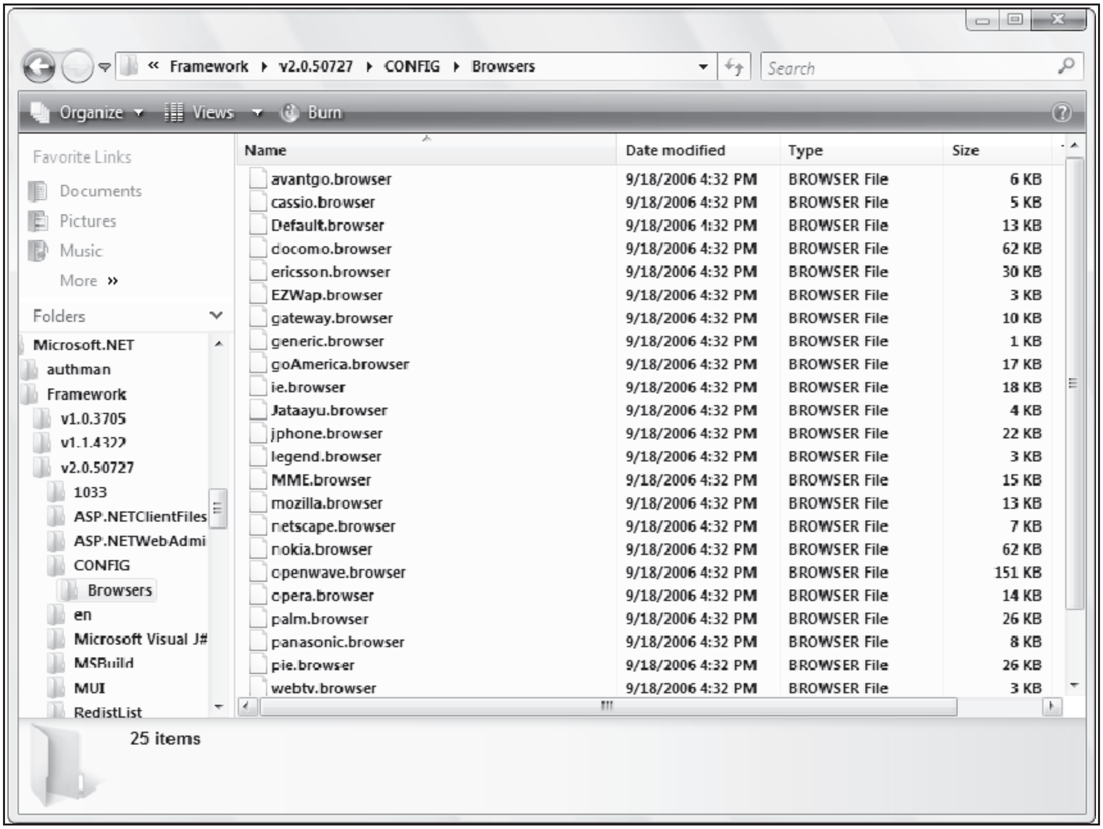
> *The* `HttpBrowserCapabilities` *class contains all the browser properties. The properties can be set and retrieved in this section. The* `<browserCaps>` *element has been deprecated since ASP.NET 2.0 and now you should instead focus on using* `.browser` *files.*

When a request is received from a browser, the browser capabilities component identifies the browser's capabilities from the request headers.

For each browser, compile a collection of settings relevant to applications. These settings may either be statically configured or gathered from request headers. Allow the application to extend or modify the capabilities settings associated with browsers and to access values through a strongly typed object model. The ASP.NET mobile capabilities depend on the browser capabilities component.

In ASP.NET 3.5, all the browser capability information is represented in browser definition files. The browser definitions are stored in `*.browser` file types and specified in XML format. A single file may contain one or more browser definitions. The `*.browser` files are stored in the `Config\Browsers` subdirectory of the Framework installation directory (for example, `[WinNT\Windows]\Microsoft.NET\Framework\v2.0.50727\CONFIG\Browsers`), as shown in Figure 31-3. Application-specific browser definition files are stored in the `/Browsers` subdirectory of the application.

> *In ASP.NET 1.0 and 1.1, the browser cap information was stored in the* `machine.config` *and* `web.config` *files themselves.*


« Framework ▸ v2.0.50727 ▸ CONFIG ▸ Browsers
Search
Organize
Views
Burn
Favorite Links
Documents
Pictures
Music
More »
Folders
Microsoft.NET
authman
Framework
v1.0.3705
v1.1.4322
v2.0.50727
1033
ASP.NETClientFiles
ASP.NETWebAdmi
CONFIG
Browsers
en
Microsoft Visual J#
MSBuild
MUI
RedistList
25 items

Name	Date modified	Type	Size
avantgo.browser	9/18/2006 4:32 PM	BROWSER File	6 KB
cassio.browser	9/18/2006 4:32 PM	BROWSER File	5 KB
Default.browser	9/18/2006 4:32 PM	BROWSER File	13 KB
docomo.browser	9/18/2006 4:32 PM	BROWSER File	62 KB
ericsson.browser	9/18/2006 4:32 PM	BROWSER File	30 KB
EZWap.browser	9/18/2006 4:32 PM	BROWSER File	3 KB
gateway.browser	9/18/2006 4:32 PM	BROWSER File	10 KB
generic.browser	9/18/2006 4:32 PM	BROWSER File	1 KB
goAmerica.browser	9/18/2006 4:32 PM	BROWSER File	17 KB
ie.browser	9/18/2006 4:32 PM	BROWSER File	18 KB
Jataayu.browser	9/18/2006 4:32 PM	BROWSER File	4 KB
jphone.browser	9/18/2006 4:32 PM	BROWSER File	22 KB
legend.browser	9/18/2006 4:32 PM	BROWSER File	3 KB
MME.browser	9/18/2006 4:32 PM	BROWSER File	15 KB
mozilla.browser	9/18/2006 4:32 PM	BROWSER File	13 KB
netscape.browser	9/18/2006 4:32 PM	BROWSER File	7 KB
nokia.browser	9/18/2006 4:32 PM	BROWSER File	62 KB
openwave.browser	9/18/2006 4:32 PM	BROWSER File	151 KB
opera.browser	9/18/2006 4:32 PM	BROWSER File	14 KB
palm.browser	9/18/2006 4:32 PM	BROWSER File	26 KB
panasonic.browser	9/18/2006 4:32 PM	BROWSER File	8 KB
pie.browser	9/18/2006 4:32 PM	BROWSER File	26 KB
webtv.browser	9/18/2006 4:32 PM	BROWSER File	3 KB

Figure 31-3

The browser definition file format defines each browser as an entity, self-contained in a `<browser>` XML element. Each browser has its own ID that describes a class of browser and its parent class. The root node of a browser definition file is the `<browsers>` element and multiple browser entries identified using the `id` attribute of the `<browser>` element.

Listing 31-15 shows a section of the `ie.browser` file.

Listing 31-15: Content of IE.browser file

```
<browsers>
    <browser id="IE" parentID="Mozilla">
        <identification>
            <userAgent match="^Mozilla[^(]*\([C|c]ompatible;\s*MSIE
         (?'version'(?'major'\d+)(?'minor'\.\d+)(?'letters'\w*))(?'extra'[^)]*)" />
            <userAgent nonMatch="Opera" />
            <userAgent nonMatch="Go\.Web" />
            <userAgent nonMatch="Windows CE" />
            <userAgent nonMatch="EudoraWeb" />
        </identification>
        <capture>
        </capture>
        <capabilities>

            <capability name="browser"             value="IE" />
            <capability name="extra"               value="${extra}" />
            <capability name="isColor"             value="true" />
            <capability name="letters"             value="${letters}" />
            <capability name="majorversion"        value="${major}" />
            <capability name="minorversion"        value="${minor}" />
            <capability name="screenBitDepth"      value="8" />
            <capability name="type"                value="IE${major}" />
            <capability name="version"             value="${version}" />

        </capabilities>
    </browser>
    ...
```

The `id` attribute of the `<browser>` element uniquely identifies the class of browser. The `parentID` attribute of the `<browser>` element specifies the unique ID of the parent browser class. Both the `id` and the `parentID` are required values.

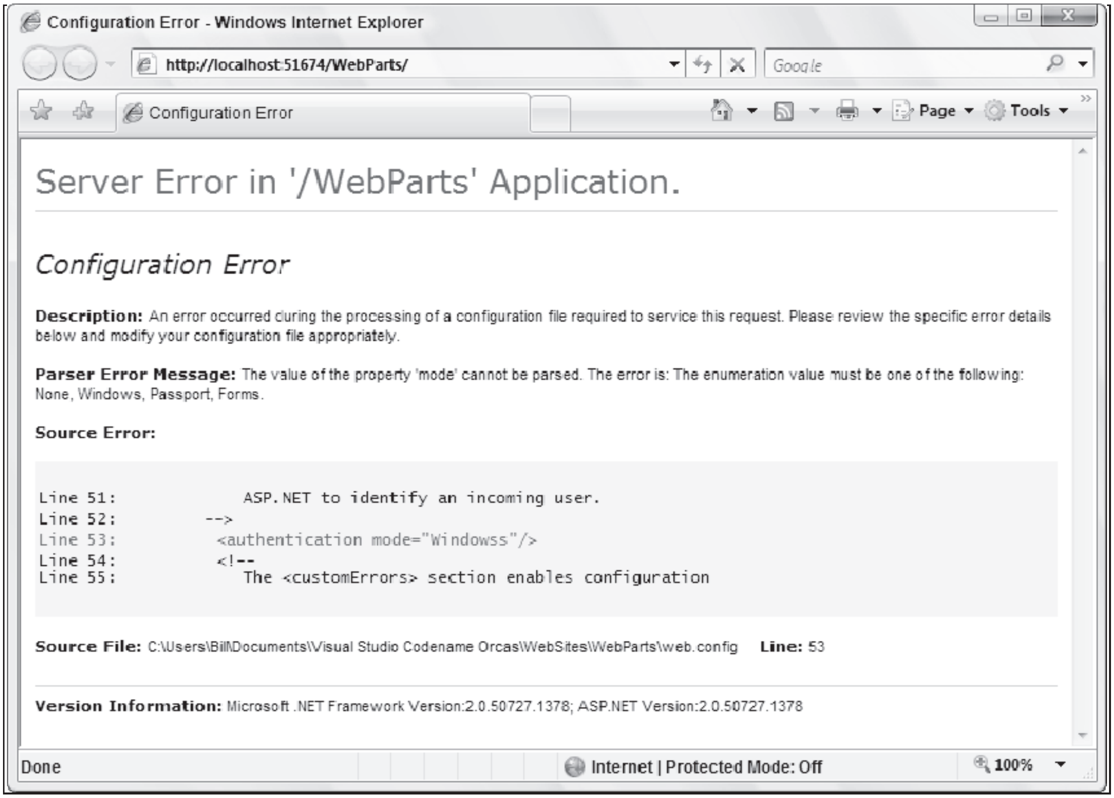
> *Before running an ASP.NET application, the framework compiles all the browser definitions into an assembly and installs the compilation in GAC. When the browser definition files at the system level are modified, they do not automatically reflect the change in each and every ASP.NET application. Therefore, it becomes the responsibility of the developer or the installation tool to update this information. You can send the updated browser information to all the ASP.NET applications by running the* `aspnet_regbrowsers.exe` *utility provided by the framework. When the* `aspnet_regbrowsers.exe` *utility is called, the browser information is recompiled and the new assembly is stored in the GAC; this assembly is reused by all the ASP.NET applications. Nevertheless, browser definitions at the application level are automatically parsed and compiled on demand when the application is started. If any changes are made to the application's* `/Browsers` *directory, the application is automatically recycled.*

Custom Errors

When the ASP.NET application fails, the ASP.NET page can show the default error page with the source code and line number of the error. However, this approach has a few problems:

- The source code and error message may not make any sense to a less experienced end user.
- If the same source code and the error messages are displayed to a hacker, subsequent damage could result.

Displaying too much error information could provide important implementation details that you are in most cases going to want to keep from the public. Figure 31-4 shows an example of this.


Configuration Error - Windows Internet Explorer
http://localhost:51674/WebParts/
Google
Configuration Error
Page
Tools
Done
Internet | Protected Mode: Off
100%

Server Error in '/WebParts' Application.

Configuration Error

Description: An error occurred during the processing of a configuration file required to service this request. Please review the specific error details below and modify your configuration file appropriately.

Parser Error Message: The value of the property 'mode' cannot be parsed. The error is: The enumeration value must be one of the following: None, Windows, Passport, Forms.

Source Error:

```
Line 51:            ASP.NET to identify an incoming user.
Line 52:        -->
Line 53:         <authentication mode="Windowss"/>
Line 54:         <!--
Line 55:            The <customErrors> section enables configuration
```

Source File: C:\Users\Bill\Documents\Visual Studio Codename Orcas\WebSites\WebParts\web.config **Line:** 53

Version Information: Microsoft .NET Framework Version:2.0.50727.1378; ASP.NET Version:2.0.50727.1378

Figure 31-4

However, ASP.NET provides excellent infrastructure to prevent this kind of error information. The `<customErrors>` section provides a means for defining custom error messages in an ASP.NET application. The syntax is as follows:

```
<customErrors defaultRedirect="[url]" mode="[on/off/remote]">
   <error statusCode="[statuscode]" redirect="[url]" />
</customErrors>
```

- `defaultRedirect`: Specifies the default URL to which the client browser should be redirected if an error occurs. This is an optional setting.
- `mode`: Specifies if the status of the custom errors is enabled, disabled, or shown only to remote machines. The possible values are `On`, `Off`, `RemoteOnly`. `On` indicates that the custom errors are enabled. `Off` indicates that the custom errors are disabled. `RemoteOnly` indicates that the custom errors are shown only to remote clients.
- `customErrors`: The `<customErrors>` section supports multiple `<error>` subelements that are used to define custom errors. Each `<error>` subelement can include a `statusCode` attribute and a URL.

Authentication

In Chapter 21, you see the authentication process in detail. In this section, you can review configuration-specific information. Authentication is a process that verifies the identity of the user and establishes the identity between the server and a request. Because HTTP is a stateless protocol, the authentication information is persisted somewhere in the client or the server; ASP.NET supports both of these.

You can store the server-side information in `Session` objects. When it comes to client side, you have many options:

- Cookies
- ViewState
- URL
- Hidden fields

ASP.NET 3.5 supports following authentication methods out of the box:

- Windows authentication
- Passport authentication
- Forms Authentication

If you would like to disable authentication, you can use the setting `mode = "None"`:

```
<authentication mode="None" />
```

Windows Authentication

ASP.NET relies on IIS's infrastructure to implement Windows authentication, and Windows authentication enables you to authenticate requests using Windows Challenge/Response semantics. When the Web server receives a request, it initially denies access to the request (which is a challenge). This triggers the browser to pop up a window to collect the credentials; the request responds with a hashed value of the Windows credentials, which the server can then choose to authenticate.

To implement Windows authentication, you configure the appropriate Web site or virtual directory using IIS. You can then use the `<authentication>` element to mark the Web application or virtual directory with Windows authentication. This is illustrated in Listing 31-16.

Listing 31-16: Setting authentication to Windows authentication

```
<configuration>
   <system.web>

     <authentication mode="Windows">

   </system.web>
</configuration>
```

The `<authentication>` *element can be declared only at the machine, site, or application level. Any attempt to declare it in a configuration file at the subdirectory or page level results in a parser error message.*

Passport Authentication

ASP.NET 3.5 relies on the Passport SDK to implement Passport authentication, which is promoted by Microsoft Corporation. Passport is a subscription-based authentication mechanism that allows end users to remember a single username/password pair across multiple Web applications that implement Passport authentication.

ASP.NET 3.5 authenticates users based on the credentials presented by users. The Passport service sends a token back to authenticate. The token is stored in a site-specific cookie after it has been authenticated with `login.passport.com`. Using the `redirectUrl` attribute of the `<passport>` authentication option, you can control how non-authenticated Passport users are directed, as in the following example:

```
<passport redirectUrl="/Passport/SignIn.aspx" />
```

Forms Authentication

Forms Authentication is the widely used authentication mechanism. Forms Authentication can be configured using the `<authentication>` section along with the `<forms>` subsection. The structure of an `<authentication>` section that deals with forms authentication in the configuration file is presented in Listing 31-17.

Listing 31-17: The <authentication> section working with forms authentication

```
<configuration>
   <system.web>

      <authentication mode="Forms">
         <forms
            name="[name]"
            loginUrl="[url]"
            protection="[All|None|Encryption|Validation]"
            timeout="30"
            path="/"
            requireSSL="[true|false]"
            slidingExpiration="[true|false]"
            cookieless="UseCookies|UseUri|AutoDetect|UseDeviceProfile"
            defaultUrl="[url]"
            domain="string">
            <credentials passwordFormat="[Clear, SHA1, MD5]">
              <user name="[UserName]" password="[password]"/>
            </credentials>
          </forms>
        </authentication>

      </system.web>
    </configuration>
```

Each attribute is shown in detail in the following list:

- `name`: Specifies the name of the HTTP authentication ticket. The default value is `.ASPXAUTH`.
- `loginUrl`: Specifies the URL to which the request is redirected if the current request doesn't have a valid authentication ticket.
- `protection`: Specifies the method used to protect cookie data. Valid values are `All`, `None`, `Encryption`, and `Validation`.
  - `Encryption`: Specifies that content of the cookie is encrypted using `TripleDES` or `DES` cryptography algorithms in the configuration file. However, the data validation is not done on the cookie.
  - `Validation`: Specifies that content of the cookie is not encrypted, but validates that the cookie data has not been altered in transit.
  - `All`: Specifies that content of the cookie is protected using both data validation and encryption. The configured data validation algorithm is used based on the `<machineKey>` element, and Triple DES is used for encryption. The default value is `All`, and it indicates the highest protection available.
  - `None`: Specifies no protection mechanism is applied on the cookie. Web applications that do not store any sensitive information and potentially use cookies for personalization can look at this option. When `None` is specified, both encryption and validation are disabled.
- `timeout`: Specifies cookie expiration time in terms of minutes. The `timeout` attribute is a sliding value, which expires `n` minutes from the time the last request was received. The default value is 30 minutes.
- `path`: Specifies the path to use for the issued cookie. The default value is `/` to avoid difficulties with mismatched case in paths because browsers are strictly case-sensitive when returning cookies.
- `requireSSL`: Specifies whether Forms Authentication should happen in a secure HTTPS connection.
- `slidingExpiration`: Specifies whether valid cookies should be updated periodically when used. When false, a ticket is good for only the duration of the period for which it is issued, and a user must re-authenticate even during an active session.
- `cookieless`: Specifies whether cookieless authentication is supported. Supported values are `UseCookies`, `UseUri`, `Auto`, and `UseDeviceProfile`. The default value is `UseDeviceProfile`.
- `defaultUrl`: Specifies the default URL used by the login control to control redirection after authentication.

- `domain`: Specifies the domain name string to be attached in the authentication cookie. This attribute is particularly useful when the same authentication cookie is shared among multiple sites across the domain.

It is strongly recommended that the `loginUrl` *should be an SSL URL (*`https://`*) to keep secure credentials secure from prying eyes.*

Anonymous Identity

Many application types require the capability to work with anonymous users, although this is especially true for e-commerce Web applications. In these cases, your site must support *both* anonymous and authenticated users. When anonymous users are browsing the site and adding items to a shopping cart, the Web application needs a way to uniquely identify these users. For example, if you look at busy e-commerce Web sites such as `Amazon.com` or `BN.com`, they do not have a concept called anonymous users. Rather these sites assign a unique identity to each user.

In ASP.NET 1.0 and 1.1, no out-of-the box feature existed to enable a developer to achieve this identification of users. Most developers used `SessionID` to identify users uniquely. They experienced a few pitfalls inherent in this method. Since the introduction of ASP.NET 2.0, ASP.NET has had anonymous identity support using the `<anonymousIdentification>` section in the configuration file. The following listing here in Listing 31-18 shows the `<anonymousIdentification>` configuration section settings.

Listing 31-18: Working with anonymous identification in the configuration file

```
<configuration>
  <system.web>

    <anonymousIdentification
               enabled="false"
               cookieName=".ASPXANONYMOUS
               cookieTimeout="100000"
               cookiePath="/"
               cookieRequireSSL="false"
               cookieSlidingExpiration = "true"
               cookieProtection = "Validation"
               cookieLess="UseCookies|UseUri|AutoDetect|UseDeviceProfile"
               domain="... ."    />

  </system.web>
</configuration>
```

The `enabled` attribute within the `<anonymousIdentification>` section specifies whether the anonymous access capabilities of ASP.NET are enabled. The other attributes are comparable to those in the authentication section from Listing 31-17. When working with anonymous identification, it is possible that the end user will have cookies disabled in their environments. When cookies are not enabled by the end user, the identity of the user is then stored in the URL string within the end user's browser.

Authorization

The authorization process verifies whether a user has the privilege to access the resource he is trying to request. ASP.NET 3.5 supports both file and URL authorization. The authorization process dictated by an application can be controlled by using the `<authorization>` section within the configuration file.

The `<authorization>` section, as presented in Listing 31-19, can contain subsections that either allow or deny permission to a user, a group of users contained within a specific role in the system, or a request that is coming to the server in a particular fashion (such as an HTTP GET request). Optionally, you can also use the `<location>` section to grant special authorization permission to only a particular folder or file within the application.

Listing 31-19: Authorization capabilities from the configuration file

```
<authorization>
   <allow users="" roles="" verbs="" />
   <deny users="" roles="" verbs="" />
</authorization>
```

URL Authorization

The URL Authorization is a service provided by `URLAuthorizationModule` (inherited from `HttpModule`) to control the access to resources such as `.aspx` files. The URL Authorization is very useful if you want to allow or deny certain parts of your ASP.NET application to certain people or roles.

For example, you may want to restrict the administration part of your ASP.NET application only to administrators and deny access to others. You can achieve this very easily with URL Authorization. URL Authorization can be configurable based on the user, the role, or HTTP verbs such as HTTP GET request or HTTP POST request.

You can configure URL Authorization in the `web.config` file with `<allow>` and `<deny>` attributes. For example, the following code (Listing 31-20) shows how you can allow the user `Bubbles` and deny the groups `Sales` and `Marketing` access to the application.

Listing 31-20: An example of allowing and denying entities from the <authorization> section

```
<system.web>
    <authorization>
        <allow users="Bubbles" />
        <deny roles="Sales, Marketing" />
    </authorization>
</system.web>
```

The `<allow>` and `<deny>` elements support `users`, `roles`, and `verbs` values. As you can see from the previous code example, you can add multiple users and groups by separating them with commas.

Two special characters, an asterisk (`*`) and a question mark (`?`), are supported by `URLAuthorizationModule`. The asterisk symbol represents all users (anonymous and registered) and the question mark represents only anonymous users. The following code example in Listing 31-21 denies access to all anonymous users and grants access to anyone contained within the Admin role.

Listing 31-21: Denying anonymous users

```
<system.web>
    <authorization>
        <allow roles="Admin" />
        <deny users="?" />
    </authorization>
</system.web>
```

You can also grant or deny users or groups access to certain HTTP methods. In the example in Listing 31-22, access to the HTTP GET method is denied to the users contained within the Admin role, whereas access to the HTTP POST method is denied to all users.

Listing 31-22: Denying users and roles by verb

```
<system.web>
    <authorization>
        <deny verbs="GET" roles="Admin" />
        <deny verbs="POST" users="*" />
    </authorization>
</system.web>
```

File Authorization

It is possible to construct the authorization section within the configuration file so that what is specified can be applied to a specific file or directory using the `<location>` element. For example, suppose you have a root directory called `Home` within your application and nested within that root directory you have a subdirectory called `Documents`. Suppose you want to allow access to the `Documents` subdirectory only to those users contained within the `Admin` role. This scenario is illustrated in Listing 31-23.

Listing 31-23: Granting access to the Documents subdirectory for the Admin role

```
<configuration>
   <location path="Documents">
      <system.web>
         <authorization>
            <allow roles="Admin" />
            <deny users="*" />
         </authorization>
      </system.web>
</location>
</configuration>
```

The ASP.NET application does not verify the path specified in the path attribute. If the given path is invalid, ASP.NET does not apply the security setting.

You can also set the security for a single file as presented in Listing 31-24.

Listing 31-24: Granting access to a specific file for the Admin role

```
<configuration>
   <location path="Documents/Default.aspx">
      <system.web>
         <authorization>
            <allow roles="Admin" />
            <deny users="*" />
         </authorization>
      </system.web>
   </location>
</configuration>
```

Locking-Down Configuration Settings

ASP.NET's configuration system is quite flexible in terms of applying configuration information to a specific application or folder. Even though the configuration system is flexible, in some cases you may want to limit the configuration options that a particular application on the server can control. For example, you could decide to change the way in which the ASP.NET session information is stored. This lock-down process can be achieved using the `<location>` attributes `allowOverride` and `allowDefinition`, as well as the `path` attribute.

Listing 31-25 illustrates this approach. A `<location>` section in this `machine.config` file identifies the path `"Default Web Site/ExampleApplication"` and allows any application to override the `<trace>` setting through the use of the `allowOverride` attribute.

Listing 31-25: Allowing a <trace> section to be overridden in a lower configuration file

```
<configuration>
    <location path="Default Web Site/ExampleApplication" allowOverride="true">
        <trace enabled="false"/>
    </location>
</configuration>
```

The trace attribute can be overridden because the `allowOverride` attribute is set to `true`. You are able to override the tracing setting in the `ExampleApplication`'s `web.config` file and enable the local `<trace>` element, thereby overriding the settings presented in Listing 31-25.

However, if you had written the attribute as `allowOverride = "false"` in the `<location>` section of the `machine.config` file, the `web.config` file for `ExampleApplication` is unable to override that specific setting.

ASP.NET Page Configuration

When an ASP.NET application has been deployed, the `<pages>` section of the configuration file enables you to control some of the default behaviors for each and every ASP.NET page. These behaviors include options such as whether you should buffer the output before sending it or whether session state should be enabled for the entire application. An example of using the `<pages>` section is presented in Listing 31-26.

Listing 31-26: Configuring the <pages> section

```
<configuration>
  <system.web>

    <pages buffer="true"
           enableSessionState="true"
           enableViewState="true"
           enableViewStateMac="false"
           autoEventWireup="true"
           smartNavigation="false"
           masterPageFile="~/ExampleApplicationMasterPage.master"
```

Continued

```
          pageBaseType="System.Web.UI.Page"
          userControlBaseType="System.Web.UI.UserControl"
          compilationMode="Auto"
          validateRequest="true" >
       <namespaces>
          <add namespace="Wrox.ExampleApplication"/>
       </namespaces>
       <controls />
       <tagMapping />
    </pages>

  </system.web>
</configuration>
```

The following list gives you the ASP.NET page configuration information elements in detail:

- `buffer`: Specifies whether the requests must be buffered on the server before it is sent it to the client.
- `enableSessionState`: Specifies whether the session state for the current ASP.NET application should be enabled. The possible values are `true`, `false`, or `readonly`. The `readonly` value means that the application can read the session values but cannot modify them.
- `enableViewState`: Specifies whether the ViewState is enabled for all the controls. If the application does not use ViewState, you can set the `value` to `false` in the application's `web.config` file.
- `autoEventWireup`: Specifies whether ASP.NET can automatically wire-up common page events such as Load or Error.
- `smartNavigation`: Smart navigation is a feature that takes advantage of IE as a client's browser to prevent the redrawing that occurs when a page is posted back to itself. Using smart navigation, the request is sent through an `IFRAME` on the client, and IE redraws only the sections of the page that have changed. By default, this option is set to `false`. When it is enabled, it is available only to Internet Explorer browsers — all other browsers get the standard behavior.
- `masterPageFile`: Identifies the master page for the current ASP.NET application. If you wish to apply the master page template to only a specific subset of pages (such as pages contained within a specific folder of your application), you can use the `<location>` element within the `web.config` file:

```
<configuration>
   <location path="ExampleApplicationAdmin">
      <system.web>
         <pages masterPageFile="~/ExampleApplicationAdminMasterPage.master" />
      </system.web>
   </location>
</configuration>
```

- `pageBaseType`: Specifies the base class for all the ASP.NET pages in the current ASP.NET application. By default, this option is set to `System.Web.UI.Page`. However, if you want all ASP.NET pages to inherit from some other base class, you can change the default via this setting.
- `userControlBaseType`: Specifies the base class for all the ASP.NET user controls in the current ASP.NET application. The default is `System.Web.UI.UserControl`. You can override the default option using this element.

- `validateRequest`: Specifies whether ASP.NET should validate all the incoming requests that are potentially dangerous like the cross-site script attack and the script injection attack. This feature provides out-of-the-box protection against cross-site scripting and script injection attacks by automatically checking all parameters in the request, ensuring that their content does not include HTML elements. For more information about this setting, visit `http://www.asp.net/faq/RequestValidation.aspx`.
- `namespaces`: Optionally, you can import a collection of assemblies that can be included in the precompilation process.
- `compilationMode`: Specifies how ASP.NET should compile the current Web application. Supported values are `Never`, `Always`, and `Auto`. When you set `compilationMode = "Never"`, this means that the pages should never be compiled. A part error occurs if the page has constructs that require compilation. When you set `compilationMode = "Always"`, this means that the pages are always compiled. When you set `compilationMode = "Auto"`, ASP.NET does not compile the pages if that is possible.

Include Files

Unlike ASP.NET 1.0 and 1.1, ASP.NET 2.0 and 3.5 support *include* files in both the `machine.config` and the `web.config` files. When configuration content is to be included in multiple places or inside the location elements, an include file is an excellent way to encapsulate the content.

Any section in a configuration file can include content from a different file using the `configSource` attribute in the `<pages>` section. The value of the attribute indicates a virtual relative filename to the include file. Listing 31-27 is an example of such a directive.

Listing 31-27: Adding additional content to the web.config file

```
<configuration>
    <system.web>
        <pages configSource="SystemWeb.config" />
    </system.web>
</configuration>
```

The configuration include files can contain information that applies to a single section, and a single include file cannot contain more than one configuration section or a portion of a section. If the `configSource` attribute is present, the section element in the source file should not contain any other attribute or any child element.

Nevertheless, the include file is not a full configuration file. It should contain only the include section, as presented in Listing 31-28.

Listing 31-28: The SystemWeb.config file

```
<pages authentication mode="Forms" />
```

The `configSource` attribute cannot be nested. An include file cannot nest another file inside it using the `configSource` attribute.

> *When an ASP.NET configuration file is changed, the application is restarted at runtime. When an external include file is used within the configuration file, the configuration reload happens without restarting the application.*

Configuring ASP.NET Runtime Settings

The general configuration settings are those that specify how long a given ASP.NET resource, such as a page, is allowed to execute before being considered timed-out. The other settings specify the maximum size of a request (in kilobytes) or whether to use fully qualified URLs in redirects. These settings can be specified using the `<httpRuntime>` section within a configuration file. The `<httpRuntime>` element is applied at the ASP.NET application at the folder level. Listing 31-29 shows the default values used in the `<httpRuntime>` section.

Listing 31-29: The <httpRuntime> section

```
<configuration>
  <system.web>

    <httpRuntime
      useFullyQualifiedRedirectUrl="false"
      enable="true"
      executionTimeout="90"
      maxRequestLength="4096"
      requestLengthDiskThreshold="512"
      appRequestQueueLimit="5000"
      minFreeThreads="8"
      minLocalRequestFreeThreads="4"
      enableKernelOutputCache="true" />

  </system.web>
</configuration>
```

Enabling and Disabling ASP.NET Applications

The `enable` attribute specifies whether the current ASP.NET application is enabled. When set to `false`, the current ASP.NET application is disabled, and all the clients trying to connect to this site receive the HTTP 404 — File Not Found exception. This value should be set only at the machine or application level. If you set this value in any other level (such as subfolder level), it is ignored. This great feature enables the administrators to bring down the application for whatever reason without starting or stopping IIS. The default value is `true`.

Outside of this setting, it is also possible to take applications offline quickly by simply placing an `App_Offline.htm` *file in the root of your application. This* `.htm` *file does not need to actually contain anything (it will not make any difference). Just having the file in the root directory causes the application domain to come down, and all requests to the application get a Page Not Found error.*

Fully Qualified Redirect URLs

The `useFullyQualifiedRedirectUrl` attribute specifies whether the client-side redirects should include the fully qualified URL. When you are programming against the mobile devices, some devices require specifying fully qualified URLs. The default value is `false`.

Request Time-Out

The `executionTimeout` setting specifies the timeout option for an ASP.NET request time-out. The value of this attribute is the amount of time in seconds during which a resource can execute before ASP.NET times the request out. The default setting is 110 seconds. If you have a particular ASP.NET page or Web service that takes longer than 110 seconds to execute, you can extend the time limit in the configuration.

Maximum Request Length

The `maxRequestLength` attribute specifies the maximum file-size upload accepted by ASP.NET runtime. For example, if the ASP.NET application is required to process huge files, it is better to change this setting. The default is 4096. This number represents kilobytes (KB or around 4 MB).

Web applications are prone to attacks these days. The attacks range from a script injection attack to a denial of service (DoS) attack. The DoS is a typical attack that bombards the Web server with requests for large files. This huge number of requests ultimately brings down the Web server. The `maxRequestLength` attribute could save you from a DoS attack by setting a restriction on the size of requests.

Buffer Uploads

In ASP.NET 1.0 or 1.1, when a HTTP post is made (either a normal ASP.NET form post, file upload, or an XMLHTTP client-side post), the entire content is buffered in memory. This works out fine for smaller posts. However, when memory-based recycling is enabled, a large post can cause the ASP.NET worker process to recycle before the upload is completed. To avoid the unnecessary worker process recycling, ASP.NET 3.5 includes a setting called `requestLengthDiskThreshold`. This setting enables an administrator to configure the file upload buffering behavior without affecting the programming model. Administrators can configure a threshold below which requests will be buffered into memory. After a request exceeds the limit, it is transparently buffered on disk and consumed from there by whatever mechanism is used to consume the data. The valid values for this setting are numbers between `1` and `Int32.MaxSize` in KB.

When file buffering is enabled, the files are uploaded to the `codegen` folder. The default path for the `codegen` folder is the following:

```
[WinNT\Windows]\Microsoft.NET\Framework\[version]\Temporary ASP.NET Files\
[ApplicationName]
```

The files are buffered using a random name in a subfolder within the `codegen` folder called `Uploads`. The location of the `codegen` folder can be configured on a per application basis using the `tempDirectory` attribute of the `<compilation>` section.

This is not a change in ASP.NET; rather it is an internal change. When an ASP.NET 1.0 or 1.1 application is migrated to the .NET Framework 2.0 or 3.5, the ASP.NET application automatically takes advantage of this feature.

Thread Management

ASP.NET runtime uses free threads available in its thread pool to fulfill requests. The `minFreeThreads` attribute indicates the number of threads that ASP.NET guarantees is available within the thread pool. The default number of threads is eight. For complex applications that require additional threads to complete processing, this simply ensures that the threads are available and that the application will not be blocked while waiting for a free thread to schedule more work. The `minLocalRequestFreeThreads` attribute controls the number of free threads dedicated for local request processing; the default is four.

Application Queue Length

The `appRequestQueueLimit` attribute specifies the maximum number of requests that ASP.NET queues for the current ASP.NET application. ASP.NET queues requests when it does not have enough free threads to process them. The `minFreeThreads` attribute specifies the number of free threads the ASP.NET application should maintain, and this setting affects the number of items stored in the queue.

> *When the number of requests queued exceeds the limit set in the* `appRequestQueueLimit` *setting, all the incoming requests are rejected and an* `HTTP 503 - Server Too Busy` *error is thrown back to the browser.*

Output Caching

The `enableKernelOutputCache` specifies whether the output caching is enabled at the IIS kernel level (`Http.sys`). At present, this setting applies only to Web servers IIS6 and higher.

Configuring the ASP.NET Worker Process

When a request for an ASP.NET page is received by IIS, it passes the request to an unmanaged DLL called `aspnet_isapi.dll`. The `aspnet_isapi.dll` further passes the request to a separate worker process, `aspnet_wp.exe` if you are working with IIS5, which runs all the ASP.NET applications. With IIS6 and higher, however, all the ASP.NET applications are run by the `w3wp.exe` process. The ASP.NET worker process can be configured using the `<processModel>` section in the `machine.config` file.

> *All the configuration sections talked about so far are read by managed code. On the other hand, the* `<processModel>` *section is read by the* `aspnet_isapi.dll` *unmanaged DLL. Because the configuration information is read by an unmanaged DLL, the changed process model information is applied to all ASP.NET applications only after an IIS restart.*

The code example in Listing 31-30 shows the default format for the `<processModel>` section.

Listing 31-30: The structure of the <processModel> element

```
<processModel
    enable="true|false"
    timeout="hrs:mins:secs|Infinite"
    idleTimeout="hrs:mins:secs|Infinite"
    shutdownTimeout="hrs:mins:secs|Infinite"
    requestLimit="num|Infinite"
    requestQueueLimit="num|Infinite"
    restartQueueLimit="num|Infinite"
    memoryLimit="percent"
    cpuMask="num"
    webGarden="true|false"
    userName="username"
    password="password"
    logLevel="All|None|Errors"
    clientConnectedCheck="hrs:mins:secs|Infinite"
    responseDeadlockInterval="hrs:mins:secs|Infinite"
    responseRestartDeadlockInterval="hrs:mins:secs|Infinite"
    comAuthenticationLevel="Default|None|Connect|Call|
      Pkt|PktIntegrity|PktPrivacy"
      comImpersonationLevel="Default|Anonymous|Identify|
      Impersonate|Delegate"
      maxWorkerThreads="num"
      maxIoThreads="num"
   />
```

The following section looks at each of these attributes in more detail:

- `enable`: Specifies whether the process model is enabled. When set to `false`, the ASP.NET applications run under IIS's process model.

  > *When ASP.NET is running under IIS6 or higher in native mode, the IIS6 or higher process model is used and most of the* `<processModel>` *section within the configuration file is simply ignored. The* `autoConfig` *and* `requestQueueLimit` *attributes are still applied in this case.*

- `timeout`: Specifies how long the worker process lives before a new worker process is created to replace the current worker process. This value can be extremely useful if a scenario exists where the application's performance starts to degrade slightly after running for several weeks, as in the case of a memory leak. Rather than your having to manually start and stop the process, ASP.NET can restart automatically. The default value is `Infinite`.
- `idleTimeout`: Specifies how long the worker process should wait before it is shut down. You can shut down the ASP.NET worker process automatically using the `idleTimeout` option. The default value is `Infinite`. You can also set this value to a time using the format, HH:MM:SS:.
- `shutdownTimeout`: Specifies how long the worker process is given to shut itself down gracefully before ASP.NET calls the `Kill` command on the process. `Kill` is a low-level command that forcefully removes the process. The default value is 5 seconds.
- `requestLimit`: Specifies when the ASP.NET worker process should be recycled after a certain number of requests are served. The default value is `Infinite`.
- `requestQueueLimit`: Instructs ASP.NET to recycle the worker process if the limit for queued requests is exceeded. The default setting is 5000.
- `memoryLimit`: Specifies how much physical memory the worker process is allowed to consume before it is considered to be misbehaving or leaking memory. The default value is 60 percent of available physical memory.
- `username` **and** `password`: By default, all ASP.NET applications are executed using the ASPNET identity. If you want an ASP.NET application to run with a different account, you can provide the username and the password pair using these attributes.
- `logLevel`: Specifies how the ASP.NET worker process logs events. The default setting is to log errors only. However, you can also disable logging by specifying `None` or you can log everything using `All`. All the log items are written to the Windows Application Event Log.
- `clientConnectedCheck`: The `clientConnectedCheck` setting enables you to check whether the client is still connected at timed intervals before performing work. The default setting is 5 seconds.
- `responseDeadlockInterval`: Specifies how frequently the deadlock check should occur. A deadlock is considered to exist when requests are queued and no responses have been sent during this interval. After a deadlock, the process is restarted. The default value is 3 minutes.

- `responseRestartDeadlockInterval`: Specifies, when a deadlock is detected by the runtime, how long the runtime should wait before restarting the process. The default value is 9 minutes.
- `comAuthenticationLevel`: Controls the level of authentication for DCOM security. The default is set to `Connect`. Other values are `Default`, `None`, `Call`, `Pkt`, `PktIntegrity`, and `PktPrivacy`.
- `comImpersonationLevel`: Controls the authentication level for COM security. The default is set to `Impersonate`. Other values are `Default`, `Anonymous`, `Identify`, and `Delegate`.
- `webGarden`: Specifies whether Web Garden mode is enabled. The default setting is `false`. A Web Garden lets you host multiple ASP.NET worker processes on a single server, thus providing the application with better hardware scalability. Web Garden mode is supported only on multiprocessor servers.
- `cpuMask`: Specifies which processors should be affinities to ASP.NET worker processes when `webGarden = "true"`. The `cpuMask` is a hexadecimal value. The default value is all processors, shown as `0xFFFFFFFF`.
- `maxWorkerThreads`: Specifies the maximum number of threads that exist within the ASP.NET worker process thread pool. The default is 20.
- `maxIoThreads`: Specifies the maximum number of I/O threads that exist within the ASP.NET worker process. The default is 20.

Running Multiple Web Sites with Multiple Versions of Framework

In the same context, multiple Web sites within the given Web server can host multiple Web sites, and each of these sites can be bound to a particular version of a .NET Framework. This is typically done using the `aspnet_regiis.exe` utility. The `aspnet_regiis.exe` utility is shipped with each version of the framework.

This utility has multiple switches. Using the `-s` switch allows you to install the current version of the .NET Framework runtime on a given Web site. Listing 31-31 shows how to install .NET Framework version 1.1 on the ExampleApplication Web site.

Listing 31-31: Installing .NET Framework version 1.1 on the ExampleApplication Web site

```
C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322>
   aspnet_regiis -s W3SVC/1ROOT/ExampleApplication
```

Storing Application-Specific Settings

Every Web application must store some application-specific information for its runtime use. The `<appSettings>` section of the `web.config` file provides a way to define custom application settings for an ASP.NET application. The section can have multiple `<add>` subelements. Its syntax is as follows:

```
<appSettings>
    <add key="[key]" value="[value]"/>
</appSettings>
```

The `<add>` subelement supports two attributes:

- `key`: Specifies the key value in an `appSettings` hash table
- `value`: Specifies the value in an `appSettings` hash table

Listing 31-32 shows how to store an application-specific connection string. The `key` value is set to `ApplicationInstanceID`, and the `value` is set to the ASP.NET application instance and the name of the server on which the application is running.

Listing 31-32: Application instance information

```
<appSettings>
    <add key="ApplicationInstanceID" value="Instance1onServerOprta"/>
</appSettings>
```

Programming Configuration Files

In ASP.NET 1.0 and 1.1 versions of the Framework provided APIs that enabled you only to read information from the configuration file. You had no way to write information into the configuration file because no out-of-the-box support was available. However, some advanced developers wrote their own APIs to write the information back to the configuration files. Because the `web.config` file is an XML file, developers were able to open configuration file using the `XmlDocument` object, modify the settings, and write it back to the disk. Even though this approach worked fine, the way to access the configuration settings were not strongly typed. Therefore, validating the values was always a challenge.

However, ASP.NET 3.5 includes APIs (ASP.NET Management Objects) to manipulate the configuration information settings in `machine.config` and `web.config` files. ASP.NET Management Objects provide a strongly typed programming model that addresses targeted administrative aspects of a .NET Web Application Server. They also govern the creation and maintenance of the ASP.NET Web configuration. Using the ASP.NET Management Objects, you can manipulate the configuration information stored in the configuration files in the local or remote computer. These can be used to script any common administrative tasks or the writing of installation scripts.

All of the ASP.NET Management Objects are stored in the `System.Configuration` and `System.Web.Configuration` namespaces. You can access the configuration using the `WebConfigurationManager` class. The `System.Configuration.Configuration` class represents a merged view of the configuration settings from the `machine.config` and hierarchical `web.config` files. The `System.Configuration` and `System.Web.Configuration` namespaces have multiple classes that enable you to access pretty much all the settings available in the configuration file. The main difference between `System.Configuration` and `System.Web.Configuration` namespaces is that the `System.Configuration` namespace contains all the classes that apply to all the .NET applications. On the other hand, the `System.Web.Configuration` namespace contains the classes that are applicable only to ASP.NET Web applications. The following table shows the important classes in `System.Configuration` and their uses.

Class Name	Purpose
`Configuration`	Enables you to manipulate the configuration stored in the local computer or a remote one.
`ConfigurationElementCollection`	Enables you to enumerate the child elements stored inside the configuration file.
`AppSettingsSection`	Enables you to manipulate the `<appSettings>` section of the configuration file.
`ConnectionStringsSettings`	Enables you to manipulate the `<connectionStrings>` section of the configuration file.

Class Name	Purpose
`ProtectedConfigurationSection`	Enables you to manipulate the `<protectedConfiguration>` section of the configuration file.
`ProtectedDataSection`	Enables you to manipulate the `<protectedData>` section of the configuration file.

The next table shows classes from the `System.Web.Configuration` and their uses.

Class Name	Purpose
`AuthenticationSection`	Enables you to manipulate the `<authentication>` section of the configuration file.
`AuthorizationSection`	Enables you to manipulate the `<authorization>` section of the configuration file.
`CompilationSection`	Enables you to manipulate the `<compilation>` section of the configuration file.
`CustomErrorsSection`	Enables you to manipulate the `<customErrors>` section of the configuration file.
`FormsAuthenticationConfiguration`	Enables you to manipulate the `<forms>` section of the configuration file.
`GlobalizationSection`	Enables you to manipulate the `<globalization>` section of the configuration file.
`HttpHandlersSection`	Enables you to manipulate the `<httpHandlers>` section of the configuration file.
`HttpModulesSection`	Enables you to manipulate the `<httpModules>` section of the configuration file.
`HttpRuntimeSection`	Enables you to manipulate the `<httpRuntime>` section of the configuration file.
`MachineKeySection`	Enables you to manipulate the `<machineKey>` section of the configuration file.
`MembershipSection`	Enables you to manipulate the `<membership>` section of the configuration file.
`PagesSection`	Enables you to manipulate the `<pages>` section of the configuration file.
`ProcessModelSection`	Enables you to manipulate the `<processModel>` section of the configuration file.
`WebPartsSection`	Enables you to manipulate the `<webParts>` section of the configuration file.

All the configuration classes are implemented based on simple object-oriented based architecture that has an entity class that holds all the data and a collection class that has methods to add, remove, enumerate, and so on. Start your configuration file programming with a simple connection string enumeration, as shown in the following section.

Enumerating Connection Strings

In a Web application, you can store multiple connection strings. Some of them are used by the system and the others may be application-specific. You can write a very simple ASP.NET application that enumerates all the connection strings stored in the `web.config` file, as shown in Listing 31-33.

Listing 31-33: The web.config file

```
<?xml version="1.0" ?>
<configuration>

    <appSettings>
        <add key="symbolServer" value="192.168.1.1" />
    </appSettings>
    <connectionStrings>
        <add name="ExampleApplication"
        connectionString="server=ExampleApplicationServer;
        database=ExampleApplicationDB;uid=WebUser;pwd=P@$$worD9"
        providerName="System.Data.SqlClient"
        />
    </connectionStrings>
    <system.web>
        <compilation debug="false" />
        <authentication mode="None" />
    </system.web>

</configuration>
```

As shown in Listing 31-33, one application setting points to the symbol server, and one connection string is stored in the `web.config` file. Use the `ConnectionStrings` collection of the `System.Web.Configuration.WebConfigurationManager` class to read the connection strings, as seen in Listing 31-34.

Listing 31-34: Enum.aspx

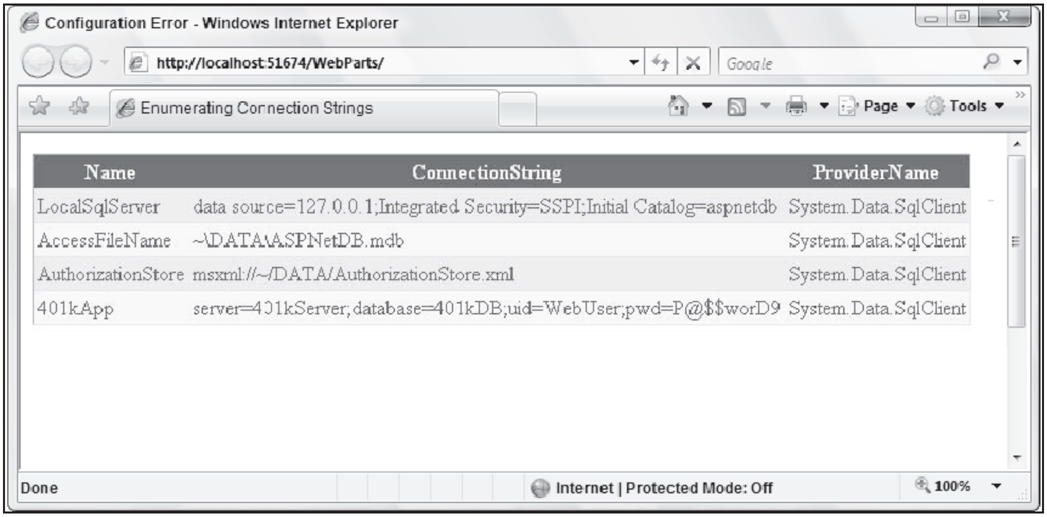
VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    GridView1.DataSource = _
        System.Web.Configuration.WebConfigurationManager.ConnectionStrings
    GridView1.DataBind()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    GridView1.DataSource =
       System.Web.Configuration.WebConfigurationManager.ConnectionStrings;
    GridView1.DataBind();
}
```

As shown in Listing 31-34, you've bound the `ConnectionStrings` property collection of the `WebConfigurationManager` class into the GridView control. The `WebConfigurationManager` class returns an instance of the `Configuration` class and the `ConnectionStrings` property is a static (shared in Visual Basic) property. Therefore, you are just binding the property collection into the GridView control. Figure 31-5 shows the list of connection strings stored in the ASP.NET application.


Configuration Error - Windows Internet Explorer
http://localhost:51674/WebParts/
Google
Enumerating Connection Strings
Page
Tools
Done
Internet | Protected Mode: Off
100%

Name	ConnectionString	ProviderName
LocalSqlServer	data source=127.0.0.1;Integrated Security=SSPI;Initial Catalog=aspnetdb	System.Data.SqlClient
AccessFileName	~\DATA\ASPNetDB.mdb	System.Data.SqlClient
AuthorizationStore	msxml://~/DATA/AuthorizationStore.xml	System.Data.SqlClient
401kApp	server=401kServer;database=401kDB;uid=WebUser;pwd=P@$$worD9	System.Data.SqlClient

Figure 31-5

Adding a connection string at runtime is also a very easy task. If you do it as shown in Listing 31-35, you get an instance of the configuration object. Then you create a new `connectionStringSettings` class. You add the new class to the collection and call the update method. Listing 31-35 shows examples of this in both VB and C#.

Listing 31-35: Adding a connection string

VB

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    ' Get the file path for the current web request
    Dim webPath As String = Request.ApplicationPath

    Try
        ' Get configuration object of the current web request
        Dim config As Configuration = _
         System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration(webPath)


        ' Create new connection setting from text boxes
        Dim newConnSetting As New _
        ConnectionStringSettings(txtName.Text, txtValue.Text, txtProvider.Text)

        ' Add the connection string to the collection
        config.ConnectionStrings.ConnectionStrings.Add(newConnSetting)
        ' Save the changes
        config.Save()
    Catch cEx As ConfigurationErrorsException
        lblStatus.Text = "Status: " + cEx.ToString()
    Catch ex As System.UnauthorizedAccessException
        ' The ASP.NET process account must have read/write access to the directory
        lblStatus.Text = "Status: " + "The ASP.NET process account must have
        read/write access to the directory"
    Catch eEx As Exception
        lblStatus.Text = "Status: " + eEx.ToString()
    End Try

    ShowConnectionStrings()
End Sub
```

C#

```
protected void Button1_Click(object sender, EventArgs e)
{
    // Get the file path for the current web request
    string webPath = Request.ApplicationPath;

    // Get configuration object of the current web request
    Configuration config =
    System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration(webPath);

    // Create new connection setting from text boxes
    ConnectionStringSettings newConnSetting = new
    ConnectionStringSettings(txtName.Text, txtValue.Text, txtProvider.Text);

    try
    {
        // Add the connection string to the collection
        config.ConnectionStrings.ConnectionStrings.Add(newConnSetting);

        // Save the changes
        config.Save();
    }
    catch (ConfigurationErrorsException cEx)
    {
        lblStatus.Text = "Status: " + cEx.ToString();
    }
    catch (System.UnauthorizedAccessException uEx)
    {
        // The ASP.NET process account must have read/write access to the directory
        lblStatus.Text = "Status: " + "The ASP.NET process account must have" +
           "read/write access to the directory";
    }
    catch (Exception eEx)
    {
        lblStatus.Text = "Status: " + eEx.ToString();
    }

    // Reload the connection strings in the list box
    ShowConnectionStrings();
}
```

Manipulating a machine.config File

The `OpenMachineConfiguration` method of the `System.Configuration.ConfigurationManager` class provides a way to manipulate the `machine.config` file. The `OpenMachineConfiguration` method is a static method.

Listing 31-36 shows a simple example that enumerates all the section groups stored in the `machine.config` file. As shown in this listing, you're getting an instance of the configuration object using the `OpenMachineConfiguration` method. Then you are binding the `SectionGroups` collection with the GridView control.

Listing 31-36: Configuration groups from machine.config

VB

```
Protected Sub Button2_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    ' List all the SectionGroups in machine.config file
    Dim configSetting As Configuration = _
      System.Configuration.ConfigurationManager.OpenMachineConfiguration()
    GridView1.DataSource = configSetting.SectionGroups
    GridView1.DataBind()
End Sub
```

C#

```
protected void Button2_Click(object sender, EventArgs e)
{
    // List all the SectionGroups in machine.config file
    Configuration configSetting =
      System.Configuration.ConfigurationManager.OpenMachineConfiguration();
    GridView1.DataSource = configSetting.SectionGroups;
    GridView1.DataBind();
}
```

In the same way, you can list all the configuration sections using the `Sections` collections, as shown in Listing 31-37.

Listing 31-37: Configuration sections from machine.config

VB

```
Protected Sub Button2_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    ' List all the SectionGroups in machine.config file
    Dim configSetting As Configuration = _
      System.Configuration.ConfigurationManager.OpenMachineConfiguration()
    GridView1.DataSource = configSetting.Sections
    GridView1.DataBind()
End Sub
```

C#

```
protected void Button2_Click(object sender, EventArgs e)
{
    // List all the SectionGroups in machine.config file
    Configuration configSetting =
      System.Configuration.ConfigurationManager.OpenMachineConfiguration();
```

```
        GridView1.DataSource = configSetting.Sections;
        GridView1.DataBind();
    }
```

Manipulating web.config from Remote Servers

The ASP.NET Management Objects also provide a way to read configuration information from remote servers.

For example, if you would like to manipulate the Expense Web application's configuration file located on the imaginary `Optra.Microsoft.com` site, you can do so as shown in Listing 31-38.

Listing 31-38: Manipulating a remote server's web.config

VB

```
' Connect to the web application Expense on Optra.Microsoft.com server
Dim configSetting As Configuration = _
   System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration _
      ("/Expense", "1", "Optra.Microsoft.com")

Dim section As System.Configuration.ConfigurationSection = _
   configSetting.GetSection("appSettings")

Dim element As KeyValueConfigurationElement = _
   CType(configSetting.AppSettings.Settings("keySection"), _
     KeyValueConfigurationElement)

If Not element Is Nothing Then
   Dim value As String = "New Value"
   element.Value = value

   Try
      config.Save()
   Catch ex As Exception
      Response.Write(ex.Message)
   End Try
End If
```

C#

```
// Connect to the web application Expense on Optra.Microsoft.com server
Configuration configSetting =
   System.Web.Configuration.WebConfigurationManager.OpenWebConfiguration
   ("/Expense", "1", "Optra.Microsoft.com");

ConfigurationSection section = configSetting.GetSection("appSettings");

KeyValueConfigurationElement element =
   configSetting.AppSettings.Settings["keySection"];

if (element != null)
{
   string value = "New Value";
   element.Value = value;
```

Continued

```
        try
        {
           configSetting.Save();
        }
           catch (Exception ex)
        {
           Response.Write(ex.Message);
        }
     }
```

The code in Listing 31-38 demonstrates how to give the machine address in the constructor method to connect to the remote server. Then you change a particular `appSettings` section to a new value and save the changes.

Protecting Configuration Settings

When ASP.NET 1.0 was introduced, all the configuration information was stored in human-readable, clear-text format. However, ASP.NET 1.1 introduced a way to store the configuration information inside the registry using the Data Protection API (or DPAPI).

For example, Listing 31-39 shows how you can store a process model section's username and password information inside the registry.

Listing 31-39: Storing the username and password in the registry and then referencing these settings in the machine.config

```
<processModel
    userName="registry:HKLM\SOFTWARE\ExampleApp\Identity\ASPNET_SETREG,userName"
    password="registry:HKLM\SOFTWARE\ExampleApp\Identity\ASPNET_SETREG,password"
/>
```

ASP.NET 1.0 also acquired this functionality as a fix. Visit the following URL for more information: `http://support.microsoft.com/default.aspx?scid=kb;en-us;329290`.

ASP.NET 3.5 includes a system for protecting sensitive data stored in the configuration system. It uses industry-standard XML encryption to encrypt specified sections of configuration that contain any sensitive data.

Developers often feel apprehensive about sticking sensitive items such as connection strings, passwords, and more in the `web.config` file. For this reason, ASP.NET makes it possible to store these items in a format that is not readable by any human or machine process without intimate knowledge of the encryption techniques and keys used in the encryption process.

One of the most encrypted items in the web.config is the `<connectionStrings>` section. Listing 31-40 shows an example of a `web.config` file with an exposed connection string.

Listing 31-40: A standard connection string exposed in the web.config file

```
<?xml version="1.0"?>

<configuration>
```

```
    <appSettings/>

    <connectionStrings>
      <add name="Northwind"
   connectionString="Server=localhost;Integrated Security=True;Database=Northwind"
        providerName="System.Data.SqlClient" />
    </connectionStrings>

  <system.web>

        <compilation debug="false" />

        <authentication mode="Forms">
           <forms name="Wrox" loginUrl="Login.aspx" path="/">
              <credentials passwordFormat="Clear">
                 <user name="BillEvjen" password="Bubbles" />
              </credentials>
           </forms>
        </authentication>

    </system.web>
</configuration>
```

In this case, you might want to encrypt this connection string to the database. To accomplish this, the install of ASP.NET provides a tool called *aspnet_regiis.exe*. You find this tool at `C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727`. To use this tool to encrypt the <connectionStrings> section, open a command prompt and navigate to the specified folder using `cd C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727`. Another option is to just open the Visual Studio 2008 Command Prompt. After you are in one of these environments, you use the syntax presented in Listing 31-41 to encrypt the `<connectionStrings>` section.

Listing 31-41: Encrypting the <connectionString> section

```
aspnet_regiis -pe "connectionString" -app "/EncryptionExample"
```

Running this bit of script produces the results presented in Figure 31-6.

Looking over the script used in the encryption process, you can see that the `-pe` command specifies the section in the `web.config` file to encrypt, whereas the `-app` command specifies which application to actually work with. If you look back at the `web.config` file and examine the encryption that occurred, you see something similar to the code in Listing 31-42.

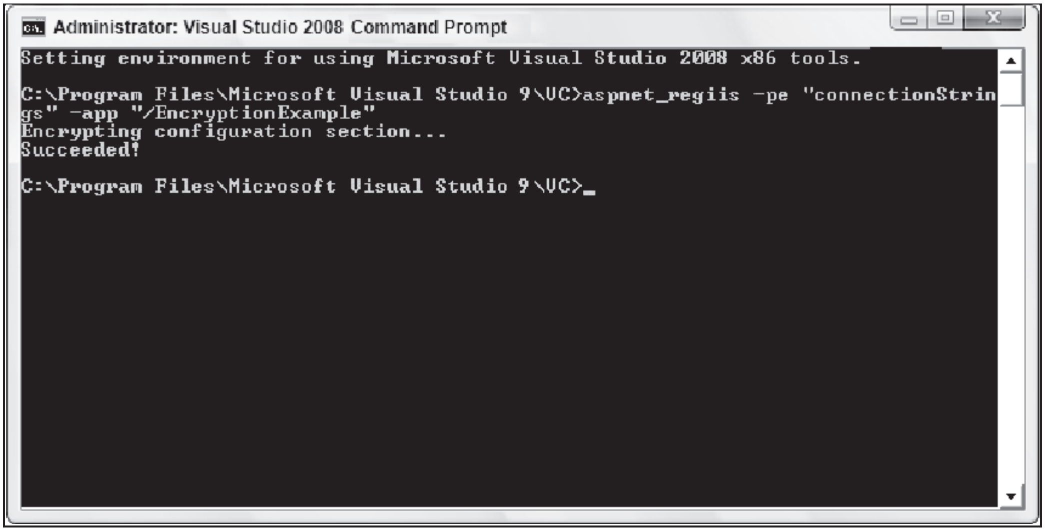
Listing 31-42: The encrypted <connectionStrings> section of the web.config

```
<?xml version="1.0"?>

<configuration>

    <appSettings/>

    <connectionStrings
```

Continued

```
     configProtectionProvider="RsaProtectedConfigurationProvider">
        <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
            xmlns="http://www.w3.org/2001/04/xmlenc#">
            <EncryptionMethod
             Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
            <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
                    <EncryptionMethod
                     Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
                    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                        <KeyName>Rsa Key</KeyName>
                    </KeyInfo>
                    <CipherData>
                        <CipherValue>
                         0s99STuGx+CdDXmWaOVc0prBFA65
                         Yub0VxDS7nOSQ79AAYcxKG7Alq1o
                         M2BqZGSmElc7c4w93qgZn0CNN
                         VHGhDLE1OjHPV942HaYhcddK5
                         5XY5j7L3WSEJFj68E2Ng9+EjU
                         o+oAGJVhCAuG8owQBaQ2Bri3+
                         tfUB/Q8LpOW4kP8=
                        </CipherValue>
                    </CipherData>
                </EncryptedKey>
            </KeyInfo>
            <CipherData>
                <CipherValue>
                 O3/PtxajkdVD/5TLGddc1/
                 C8cg8RFYl8MiRXh71h4ls=
                </CipherValue>
            </CipherData>
        </EncryptedData>
    </connectionStrings>

  <system.web>

        <compilation debug="false" />

        <authentication mode="Forms">
           <forms name="Wrox" loginUrl="Login.aspx" path="/">
              <credentials passwordFormat="Clear">
                 <user name="BillEvjen" password="Bubbles" />
              </credentials>
           </forms>
        </authentication>


    </system.web>
</configuration>
```

Now when you work with a connection string in your ASP.NET application, ASP.NET itself automatically decrypts this section in order to utilize the values stored. Looking at the `web.config` file, you can see a subsection within the `<system.web>` section that exposes a username and password as clear text. This is also something that you might want to encrypt in order to keep it away from prying eyes. Because it is a subsection, you use the script presented in Listing 31-43.



```
Setting environment for using Microsoft Visual Studio 2008 x86 tools.

C:\Program Files\Microsoft Visual Studio 9\VC>aspnet_regiis -pe "connectionStrin
gs" -app "/EncryptionExample"
Encrypting configuration section...
Succeeded!

C:\Program Files\Microsoft Visual Studio 9\VC>_
```

Figure 31-6

Listing 31-43: Encrypting the <authentication> section

```
aspnet_regiis -pe "system.web/authentication" -app "/EncryptionExample"
```

This code gives you the partial results presented in Listing 31-44.

Listing 31-44: The encrypted <authentication> section of the web.config

```
<authentication configProtectionProvider="RsaProtectedConfigurationProvider">
   <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
    xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod
       Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
          <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
             <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
                <EncryptionMethod
                 Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
                <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                   <KeyName>Rsa Key</KeyName>
                </KeyInfo>
                <CipherData>
                   <CipherValue>
                    GzTlMc89r3ees9EoMedFQrLo3FI5p3JJ9DMONWe
                    ASIww89UADkihLpmzCUPa3YtiCfKXpodr3Xt3RI
                    4zpveulZs5gIZUoX8aCl48U89dajudJn7eoJqai
                    m6wuXTGI5XrUWTgYdELCcFCloW1c+eGMRBZpNi9
                    cir4xkkh2SsHBDE=
                   </CipherValue>
                </CipherData>
             </EncryptedKey>
```

Continued

```
      </KeyInfo>
      <CipherData>
         <CipherValue>
          3MpRm+Xs+x5YsndH20lZau8/t+3RuaGv5+nTFoRXaV
          tweKdgrAVeB+PXnTjydq/u4LBXKMKmHzaBtxrqEHRD
          mNZgigLWVtfIRQ6P8cgBwtdIFhFmjm3B4tg/rA8dpJ
          ivDav2kDPp+SZ6yZ9LJzhBIe9TdJvwBQ9gJTGNVRft
          QOvdvH8c4KwYfiwZa9WCqys9WOZmw6g1a5jdW3hM//
          jiMizY1MwCECVh+T+y+f/vpP0xCkoKT9GGgHRMMrQd
          PqHUd5s7rUYp1ijQgrh1oPIXr6mx/XtzdXV8bQiEsg
          CLhsqphoVVwxkvmUKEmDQdOzdrB4sqmKgoHR3wCPyB
          npH58g==
         </CipherValue>
      </CipherData>
   </EncryptedData>
</authentication>
```

After you have sections of your `web.config` file encrypted, you need a process to decrypt these sections to their original unencrypted values. To accomplish this task, you use the aspnet_regiis tool illustrated in Listing 31-45.

Listing 31-45: Decrypting the <connectionStrings> section in the web.config file

```
aspnet_regiis -pd "connectionString" -app "/EncryptionExample"
```

Running this script returns the encrypted values to original values.

Editing Configuration Files

So far in this chapter, you have learned about configuration files and what each configuration entry means. Even though the configuration entries are in an easy, human-readable XML format, editing these entries can be cumbersome. To help with editing, Microsoft ships three tools:

- Visual Studio 2008 IDE
- Web Site Administration Tool
- ASP.NET Snap-In for IIS 6.0 or Windows Vista's Internet Information Services (IIS) Manager

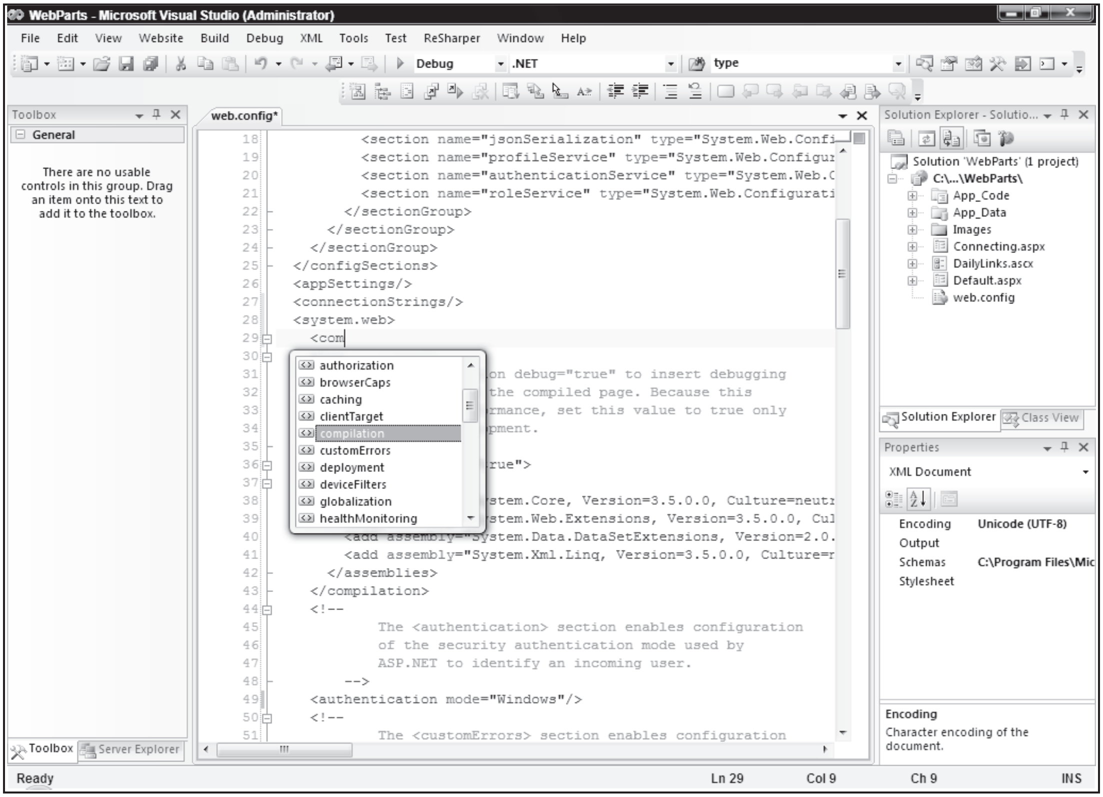
One of the nice capabilities of the Visual Studio 2008 IDE is that it supports IntelliSense-based editing for configuration files, as shown in Figure 31-7.

The Visual Studio 2008 IDE also supports XML element syntax checking, as shown in Figure 31-8.

> *XML element syntax checking and IntelliSense for XML elements are accomplished using the XSD-based XML validation feature available for all the XML files inside Visual Studio 2008. The configuration XSD file is located at* `<drive>:\Program Files\Microsoft Visual Studio 9.0\Xml\Schemas\DotNetConfig.xsd`.

The Visual Studio 2008 IDE also adds two new useful features via the XML toolbar options that can help you with formatting the configuration settings:

- **Reformat Selection:** This option reformats the current XML notes content.
- **Format the whole document:** This option formats the entire XML document.


WebParts - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug XML Tools Test ReSharper Window Help
Debug
.NET
type
Toolbox
General
web.config*
authorization
browserCaps
caching
clientTarget
compilation
customErrors
deployment
deviceFilters
globalization
healthMonitoring
Solution Explorer - Solutio...
Solution 'WebParts' (1 project)
C:\...\WebParts\
App_Code
App_Data
Images
Connecting.aspx
DailyLinks.ascx
Default.aspx
web.config
Solution Explorer
Class View
Properties
XML Document
Toolbox
Server Explorer
Ready
Ln 29
Col 9
Ch 9
INS

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

```
          <section name="jsonSerialization" type="System.Web.Confi
          <section name="profileService" type="System.Web.Configur
          <section name="authenticationService" type="System.Web.C
          <section name="roleService" type="System.Web.Configurati
        </sectionGroup>
      </sectionGroup>
    </sectionGroup>
  </configSections>
  <appSettings/>
  <connectionStrings/>
  <system.web>
    <com

                         on debug="true" to insert debugging
                         the compiled page. Because this
                         rmance, set this value to true only
                         pment.

                         rue">

                         stem.Core, Version=3.5.0.0, Culture=neutr
                         stem.Web.Extensions, Version=3.5.0.0, Cul
        <add assembly="System.Data.DataSetExtensions, Version=2.0.
        <add assembly="System.Xml.Linq, Version=3.5.0.0, Culture=r
      </assemblies>
    </compilation>
    <!--
            The <authentication> section enables configuration
            of the security authentication mode used by
            ASP.NET to identify an incoming user.
        -->
    <authentication mode="Windows"/>
    <!--
            The <customErrors> section enables configuration
```

Encoding Unicode (UTF-8)
Output
Schemas C:\Program Files\Mic
Stylesheet

Encoding
Character encoding of the document.

Figure 31-7

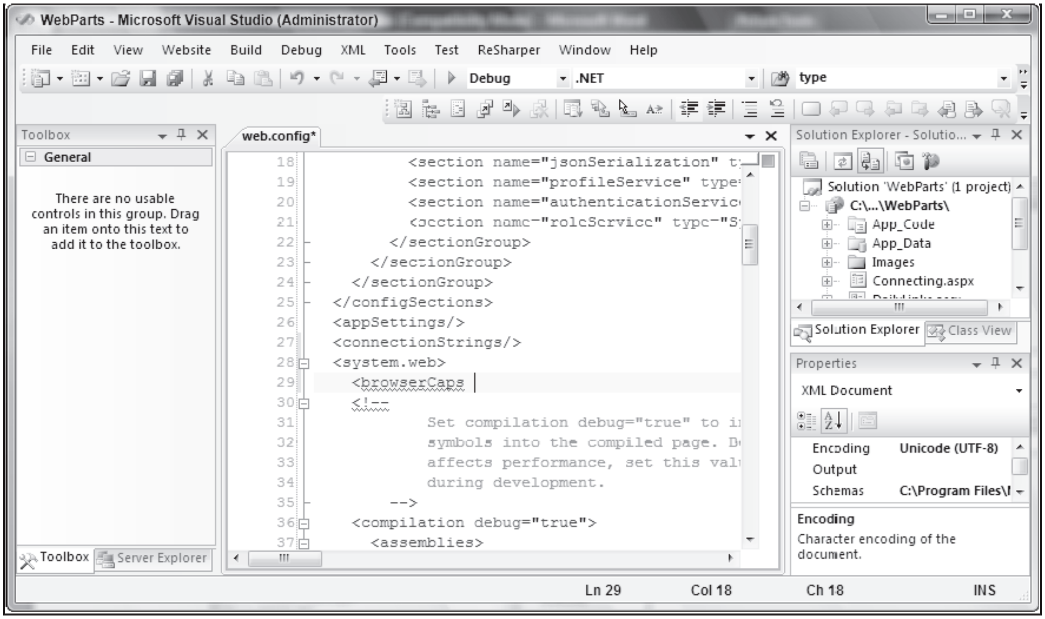
The Web Site Administration Tool and the ASP.NET Snap-In for IIS 6.0 or Window Vista's IIS Manager allow you to edit the configuration entries without knowing the XML element names and their corresponding values. Chapter 24 covers these tools in more detail.

Creating Custom Sections

In addition to using the `web.config` file as discussed, you can also extend it and add your own custom sections to the file that your can make use of just as the other sections.

One way of creating custom sections is to use some built-in handlers that enable you to read key-value pairs from the `.config` file. All three of the following handlers are from the `System.Configuration` namespace:

- ❑ `NameValueFileSectionHandler`: This handler works with the current `<appSettings>` section of the `web.config` file. You are able to use this handler to create new sections of the configuration file that behave in the same manner as the `<appSettings>` section.
- ❑ `DictionarySectionHandler`: This handler works with a dictionary collection of key-value pairs.
- ❑ `SingleTagSectionHandler`: This handler works from a single element in the configuration file and allows you to read key-value pairs that are contained as attributes and values.


WebParts - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug XML Tools Test ReSharper Window Help
Debug
.NET
type
Toolbox
General
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
Toolbox
Server Explorer
web.config*
Solution Explorer - Solutio...
Solution 'WebParts' (1 project)
C:\...\WebParts\
App_Code
App_Data
Images
Connecting.aspx
Solution Explorer
Class View
Properties
XML Document
Encoding
Character encoding of the document.
Ln 29
Col 18
Ch 18
INS

```
            <section name="jsonSerialization" t
            <section name="profileService" type
            <section name="authenticationServic
            <section name="roleService" type="S
          </sectionGroup>
        </sectionGroup>
      </sectionGroup>
    </configSections>
    <appSettings/>
    <connectionStrings/>
    <system.web>
      <browserCaps 
      <!--
              Set compilation debug="true" to i
              symbols into the compiled page. D
              affects performance, set this val
              during development.
          -->
      <compilation debug="true">
        <assemblies>
```

Encoding Unicode (UTF-8)
Output
Schemas C:\Program Files\

Figure 31-8

Next, this chapter looks at each of these handlers and some programmatic ways to customize the configuration file.

Using the NameValueFileSectionHandler Object

If you are looking to create a custom section that behaves like the `<appSettings>` section of the `web.config` file, then using this handler is the way to go. Above the `<system.web>` section of the `web.config` file, make a reference to the `NameValueFileSectionHandler` object, along with the other default references you will find in an ASP.NET 3.5 application. This additional reference is shown in Listing 31-46.

Listing 31-46: Creating your own custom section of key-value pairs in the web.config

```
<configSections>
   <section name="MyCompanyAppSettings"
    type="System.Configuration.NameValueFileSectionHandler, System,
       Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
    restartOnExternalChanges="false" />
   <sectionGroup name="system.web.extensions"
    type="System.Web.Configuration.SystemWebExtensionsSectionGroup,
       System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
       PublicKeyToken=31BF3856AD364E35">
      <sectionGroup name="scripting"
       type="System.Web.Configuration.ScriptingSectionGroup, System.Web.Extensions,
          Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">
         <section name="scriptResourceHandler"
          type="System.Web.Configuration.ScriptingScriptResourceHandlerSection,
             System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
             PublicKeyToken=31BF3856AD364E35"
```

```
            requirePermission="false" allowDefinition="MachineToApplication"/>
           <sectionGroup name="webServices"
            type="System.Web.Configuration.ScriptingWebServicesSectionGroup,
               System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
               PublicKeyToken=31BF3856AD364E35">
              <section name="jsonSerialization"
               type="System.Web.Configuration.ScriptingJsonSerializationSection,
                  System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
                  PublicKeyToken=31BF3856AD364E35"
               requirePermission="false" allowDefinition="Everywhere"/>
              <section name="profileService"
               type="System.Web.Configuration.ScriptingProfileServiceSection,
                  System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
                  PublicKeyToken=31BF3856AD364E35"
               requirePermission="false" allowDefinition="MachineToApplication"/>
              <section name="authenticationService"
               type="System.Web.Configuration.ScriptingAuthenticationServiceSection,
                  System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
                  PublicKeyToken=31BF3856AD364E35"
               requirePermission="false" allowDefinition="MachineToApplication"/>
              <section name="roleService"
               type="System.Web.Configuration.ScriptingRoleServiceSection,
                  System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
                  PublicKeyToken=31BF3856AD364E35"
               requirePermission="false" allowDefinition="MachineToApplication"/>
           </sectionGroup>
        </sectionGroup>
     </sectionGroup>
  </configSections>
```

Once you have made this reference to the `System.Configuration.NameValueFileSectionHandler` object and have given it a name (in this case, `MyCompanyAppSettings`), then you can create a section in your `web.config` that makes use of this reference. This is illustrated in Listing 31-47.

Listing 31-47: Creating your own custom key-value pair section in the web.config

```
<configuration>

   <MyCompanyAppSettings>
      <add key="Key1" value="This is value 1" />
      <add key="Key2" value="This is value 2" />
   </MyCompanyAppSettings>

   <system.web>

      <!-- Removed for clarity -->

</system.web>

</configuration>
```

After you have this in place within your `web.config` file, you can then programmatically get access to this section, as illustrated in Listing 31-48.

Listing 31-48: Getting access to your custom section in the web.config file

VB

```
Dim nvc As NameValueCollection = New NameValueCollection()
nvc = System.Configuration.ConfigurationManager.GetSection("MyCompanyAppSettings")

Response.Write(nvc("Key1") + "<br />")
Response.Write(nvc("Key2"))
```

C#

```
NameValueCollection nvc = new NameValueCollection();
nvc = ConfigurationManager.GetSection("MyCompanyAppSettings") as
   NameValueCollection;

Response.Write(nvc["Key1"] + "<br />");
Response.Write(nvc["Key2"]);
```

For this to work, you are going to have to import the `System.Collections.Specialized` namespace into the file, as this is where you will find the `NameValueCollection` object.

Using the DictionarySectionHandler Object

The `DictionarySectionHandler` works nearly the same as the `NameValueFileSectionHandler`. The difference, however, is that the `DictionarySectionHandler` returns a `HashTable` object instead of returning an `Object`.

This handler is presented in Listing 31-49.

Listing 31-49: Making a reference to the DictionarySectionHandler object

```
<configSections>
   <section name="MyCompanyAppSettings"
    type="System.Configuration.DictionarySectionHandler, System,
       Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
    restartOnExternalChanges="false" />

   <!-- Removed for clarity -->

</configSections>
```

Once in place, you can then make the same `MyCompanyAppSettings` section in the `web.config` file, as shown in Listing 31-50.

Listing 31-50: Creating your own custom key-value pair section in the web.config

```
<configuration>

   <MyCompanyAppSettings>
      <add key="Key1" value="This is value 1" />
      <add key="Key2" value="This is value 2" />
      </MyCompanyAppSettings>

      <system.web>

         <!-- Removed for clarity -->

      </system.web>

   </configuration>
```

Now that the `web.config` file is ready, you can call the items from code using the `Configuration` API, as illustrated in Listing 31-51.

Listing 31-51: Getting access to your custom section in the web.config file

VB

```
Dim ht As Hashtable = New Hashtable()
ht = System.Configuration.ConfigurationManager.GetSection("MyCompanyAppSettings")

Response.Write(ht("Key1") + "<br />")
Response.Write(ht("Key2"))
```

C#

```
Hashtable ht = new Hashtable();
ht = ConfigurationManager.GetSection("MyCompanyAppSettings") as
   Hashtable;

Response.Write(ht["Key1"] + "<br />");
Response.Write(ht["Key2"]);
```

Using the SingleTagSectionHandler Object

The `SingleTagSectionHandler` works almost the same as the previous `NameValueFileSectionHandler` and `DictionarySectionHandler`. However, this object looks to work with a single element that contains the key-value pairs as attributes.

This handler is presented in Listing 31-52.

Listing 31-52: Making a reference to the DictionarySectionHandler object

```
<configSections>
   <section name="MyCompanyAppSettings"
    type="System.Configuration.SingleTagSectionHandler, System,
       Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
    restartOnExternalChanges="false" />

   <!-- Removed for clarity -->

</configSections>
```

Once in place, you can make a different `MyCompanyAppSettings` section in the `web.config` file, as presented in Listing 31-53.

Listing 31-53: Creating your own custom key-value pair section in the web.config

```
<configuration>

 <MyCompanyAppSettings Key1="This is value 1" Key2="This is value 2" />

   <system.web>

      <!-- Removed for clarity -->

   </system.web>

</configuration>
```

Now that the `web.config` file is complete, you can call the items from code using the `Configuration` API, as illustrated in Listing 31-54.

Listing 31-54: Getting access to your custom section in the web.config file

VB

```
Dim ht As Hashtable = New Hashtable()
ht = System.Configuration.ConfigurationManager.GetSection("MyCompanyAppSettings")

Response.Write(ht("Key1") + "<br />")
Response.Write(ht("Key2"))
```

C#

```
Hashtable ht = new Hashtable();
ht = ConfigurationManager.GetSection("MyCompanyAppSettings") as
   Hashtable;

Response.Write(ht["Key1"] + "<br />");
Response.Write(ht["Key2"]);
```

Using Your Own Custom Configuration Handler

You can also create your own custom configuration handler. To do this, you first need to create a class that represents your section in the `web.config` file. In your App_Code folder, create a class called `MyCompanySettings`. This class is presented in Listing 31-55.

Listing 31-55: The MyCompanySettings class

VB

```
Public Class MyCompanySettings
    Inherits ConfigurationSection

    <ConfigurationProperty("Key1", DefaultValue:="This is the value of Key 1", _
       IsRequired:=False)> _
    Public ReadOnly Property Key1() As String
        Get
            Return MyBase.Item("Key1").ToString()
        End Get
    End Property

    <ConfigurationProperty("Key2", IsRequired:=True)> _
    Public ReadOnly Property Key2() As String
        Get
            Return MyBase.Item("Key2").ToString()
        End Get
    End Property

End Class
```

C#

```
using System.Configuration;

public class MyCompanySettings : ConfigurationSection
{
    [ConfigurationProperty("Key1", DefaultValue = "This is the value of Key 1",
     IsRequired = false)]
    public string Key1
    {
        get
        {
            return this["Key1"] as string;
        }
    }

    [ConfigurationProperty("Key2", IsRequired = true)]
    public string Key2
    {
        get
        {
            return this["Key2"] as string;
        }
    }
}
```

You can see that this class inherits from the `ConfigurationSection` and the two properties that are created using the `ConfigurationProperty` attribute. You can use a couple of attributes here, such as the `DefaultValue`, `IsRequired`, `IsKey`, and `IsDefaultCollection`.

Once you have this class in place, you can configure your application to use this handler, as illustrated in Listing 31-56.

Listing 31-56: Making a reference to the MyCompanySettings object

```
<configSections>

   <section name="MyCompanySettings" type="MyCompanySettings" />

   <!-- Removed for clarity -->

</configSections>
```

You can now use this section in your `web.config` file, as illustrated in Listing 31-57.

Listing 31-57: Creating your own custom key-value pair section in the web.config

```
<configuration>

   <MyCompanySettings Key2="Here is a value for Key2" />

   <system.web>

      <!-- Removed for clarity -->

   </system.web>

</configuration>
```

From there, you can programmatically access this from code, as illustrated in Listing 31-58.

Listing 31-58: Getting access to your custom section in the web.config file

VB

```
Dim cs As MyCompanySettings = New MyCompanySettings()
cs = ConfigurationManager.GetSection("MyCompanySettings")

Response.Write(cs.Key1 + "<br />")
Response.Write(cs.Key2)
```

C#

```
MyCompanySettings cs = ConfigurationManager.GetSection("MyCompanySettings") as
   MyCompanySettings;

Response.Write(cs.Key1 + "<br />");
Response.Write(cs.Key2);
```

Summary

In this chapter, you have seen the ASP.NET configuration system and learned how it does not rely on the IIS metabase. Instead, ASP.NET uses an XML configuration system that is human-readable.

You also looked at the two different ASP.NET XML configuration files:

- ☐ `machine.config`
- ☐ `web.config`

The `machine.config` file applies default settings to all Web applications on the server. However, if the server has multiple versions of the framework installed, the `machine.config` file applies to a particular framework version. On the other hand, a particular Web application can customize or override its own configuration information using `web.config` files. Using a `web.config` file, you can also configure the applications on an application-by-application or folder-by-folder basis.

Next, you looked at some typical configuration settings that can be applied to an ASP.NET application, such as configuring connecting strings, session state, browser capabilities, and so on. Then you looked at an overview of new ASP.NET Admin Objects and learned how to program configuration files. Finally, you learned how to protect the configuration section using cryptographic algorithms.

32

Instrumentation

Many ASP.NET developers do more than just build an application and walk away. They definitely think about how the application will behave after it is deployed. *Instrumentation* is the task that developers undertake to measure and monitor their ASP.NET applications. Depending on the situation, some instrumentation operations occur at design time, whereas others are ongoing processes that begin at runtime.

ASP.NET 3.5 gives you greater capability to apply instrumentation techniques to your applications. You will find that the ASP.NET framework includes a large series of performance counters, the capability to work with the Windows Event Tracing system, possibilities for application tracing (covered in Chapter 24), and the most exciting part of this discussion — a health monitoring system that allows you to log a number of different events over an application's lifetime.

You can monitor a deployed application in several ways. First, you learn how to work with the Windows event log.

Working with the Event Log

When working with Visual Studio 2008, you can use the event log in the Server Explorer of the IDE in a couple of different ways. You can get to the event log section in the Server Explorer by expanding the view of the server you want to work with (by clicking the plus sign next to the server) until you see the Event Logs section. You also have the option to right-click the Event Logs node and select Launch Event Viewer from the list of available options. This selection displays the same Event Viewer you are familiar with. From the Event Viewer or from the Server Explorer you can work directly with events recorded to your system.

The other option available from the Server Explorer is to expand the Event Logs node of the tree view in the Server Explorer so that you can see the additional nodes such as Application, Security, and System. If you are on Windows Vista, you will see a series of additional event categories. Expanding any of these sub-nodes allows you to see all the events that have been registered in the event log. These events are arranged by type, which makes browsing for specific events rather easy, as illustrated in Figure 32-1.

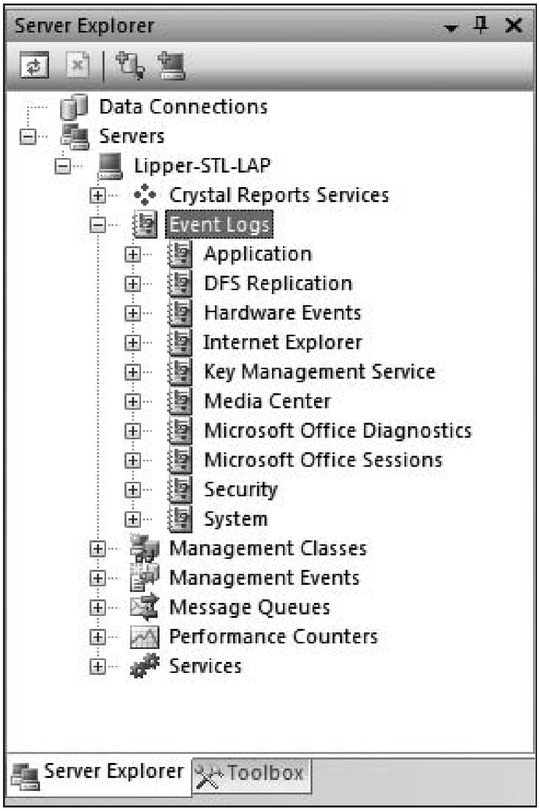

Server Explorer
Data Connections
Servers
Lipper-STL-LAP
Crystal Reports Services
Event Logs
Application
DFS Replication
Hardware Events
Internet Explorer
Key Management Service
Media Center
Microsoft Office Diagnostics
Microsoft Office Sessions
Security
System
Management Classes
Management Events
Message Queues
Performance Counters
Services
Server Explorer
Toolbox

Figure 32-1

Reading from the Event Log

It is possible to read and to write events to and from the event log from your .NET application. If you are interested in reading events from the event log, you can do so rather simply by using the `EventLog` object that is provided by .NET in the `System.Diagnostics` namespace.

To see an example of using this object, you can create an ASP.NET page that displays all the entries contained within a specified event log. To create it, you need a DropDownList control, a Button control, and a GridView control. In Visual Studio .NET 2002 and 2003, a Components tab was located within the Toolbox, and you could simply drag and drop the Event Log component onto your design surface in order to start working with it. In Visual Studio 2005, you won't find this tab within the Toolbox, but it still isn't too hard to find the objects you need. Listing 32-1 shows the simple ASP.NET page that enables you to easily display the contents of the event log.

Listing 32-1: Displaying the contents of the event logs within the browser

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Diagnostics" %>

<script runat="server">
```

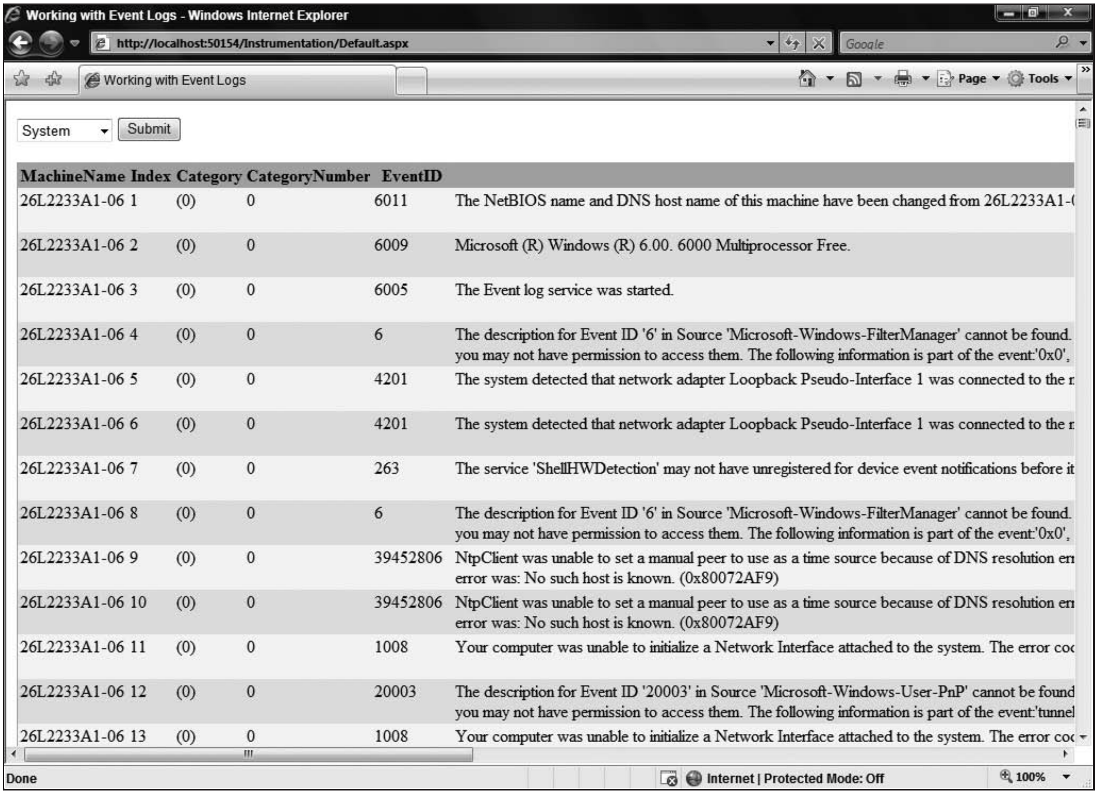
```
    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)

        Dim el As EventLog = New EventLog()
        el.Source = DropDownList1.SelectedItem.Text

        GridView1.DataSource = el.Entries
        GridView1.DataBind()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Working with Event Logs</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:DropDownList ID="DropDownList1" runat="server">
            <asp:ListItem>Application</asp:ListItem>
            <asp:ListItem>Security</asp:ListItem>
            <asp:ListItem>System</asp:ListItem>
        </asp:DropDownList>
        <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
         Text="Submit" /><br />
        <br />
        <asp:GridView ID="GridView1" runat="server"
         BackColor="LightGoldenrodYellow" BorderColor="Tan"
            BorderWidth="1px" CellPadding="2" ForeColor="Black" GridLines="None">
            <FooterStyle BackColor="Tan" />
            <SelectedRowStyle BackColor="DarkSlateBlue" ForeColor="GhostWhite" />
            <PagerStyle BackColor="PaleGoldenrod" ForeColor="DarkSlateBlue"
             HorizontalAlign="Center" />
            <HeaderStyle BackColor="Tan" Font-Bold="True" />
            <AlternatingRowStyle BackColor="PaleGoldenrod" />
            <RowStyle VerticalAlign="Top" />
        </asp:GridView>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Diagnostics" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        EventLog el = new EventLog();
        el.Source = DropDownList1.SelectedItem.Text;

        GridView1.DataSource = el.Entries;
        GridView1.DataBind();
    }
</script>
```

> *Note that you are going to have to run this with credentials that have administrative rights for this code sample to work.*

For this code to work, you import the `System.Diagnostics` namespace if you are not interested in fully qualifying your declarations. After you do so, you can create an instance of the `EventLog` object to give you access to the `Source` property. In assigning a value to this property, you use the `SelectedItem.Text` property from the DropDownList control on the page. Next, you can provide all the `EventLog` entries as the data source value to the GridView control. Finally, you bind this data source to the GridView. In the end, you get something similar to the results illustrated in Figure 32-2.


Working with Event Logs - Windows Internet Explorer
http://localhost:50154/Instrumentation/Default.aspx
Google
Working with Event Logs
Page
Tools
System
Submit
Done
Internet | Protected Mode: Off
100%

MachineName	Index	Category	CategoryNumber	EventID	
26L2233A1-06	1	(0)	0	6011	The NetBIOS name and DNS host name of this machine have been changed from 26L2233A1-
26L2233A1-06	2	(0)	0	6009	Microsoft (R) Windows (R) 6.00. 6000 Multiprocessor Free.
26L2233A1-06	3	(0)	0	6005	The Event log service was started.
26L2233A1-06	4	(0)	0	6	The description for Event ID '6' in Source 'Microsoft-Windows-FilterManager' cannot be found. you may not have permission to access them. The following information is part of the event:'0x0',
26L2233A1-06	5	(0)	0	4201	The system detected that network adapter Loopback Pseudo-Interface 1 was connected to the r
26L2233A1-06	6	(0)	0	4201	The system detected that network adapter Loopback Pseudo-Interface 1 was connected to the r
26L2233A1-06	7	(0)	0	263	The service 'ShellHWDetection' may not have unregistered for device event notifications before it
26L2233A1-06	8	(0)	0	6	The description for Event ID '6' in Source 'Microsoft-Windows-FilterManager' cannot be found. you may not have permission to access them. The following information is part of the event:'0x0',
26L2233A1-06	9	(0)	0	39452806	NtpClient was unable to set a manual peer to use as a time source because of DNS resolution er error was: No such host is known. (0x80072AF9)
26L2233A1-06	10	(0)	0	39452806	NtpClient was unable to set a manual peer to use as a time source because of DNS resolution er error was: No such host is known. (0x80072AF9)
26L2233A1-06	11	(0)	0	1008	Your computer was unable to initialize a Network Interface attached to the system. The error co
26L2233A1-06	12	(0)	0	20003	The description for Event ID '20003' in Source 'Microsoft-Windows-User-PnP' cannot be found you may not have permission to access them. The following information is part of the event:'tunnel
26L2233A1-06	13	(0)	0	1008	Your computer was unable to initialize a Network Interface attached to the system. The error co

Figure 32-2

As you can see, it is simple to pull all the events from the event log and write them to the browser screen. The next section looks at writing values back to the event log.

Writing to the Event Logs

Not only can you read from the event logs, but you can write to them as well. This can be quite handy if you want to record specific events in an event log. The following table provides a short description of the main event logs available to you. Windows Vista has other event categories, but these are the main event categories that you will use.

Event Log	Description
Application	Enables you to record application-specific events, including whether a certain event was fired, a page was loaded, or a customer made a specific purchase
Security	Enables you to track security-related issues such as security changes and breaches
System	Enables you to track system-specific items such as issues that arise with components or drivers

From your code, you can write to any of the event logs defined in the preceding table as well as to any custom event logs that you might create. To accomplish this task, create an example ASP.NET page that contains a multiline TextBox control and a Button control. On the `Button1_Click` event, you register the text you want placed in the text box directly into the Application event log. Your ASP.NET page should be similar to what is presented in Listing 32-2.

Listing 32-2: Writing to the Application event log

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Diagnostics" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, _
      ByVal e As System.EventArgs)

        If Not EventLog.SourceExists("My Application") Then
            EventLog.CreateEventSource("My Application", "Application")
        End If
    End Sub

    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)

        Dim el As EventLog = New EventLog()
        el.Source = "My Application"
        el.WriteEntry(TextBox1.Text)

        Label1.Text = "ENTERED: " & TextBox1.Text
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Working with Event Logs</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" runat="server" Height="75px"
```

Continued

```
             TextMode="MultiLine" Width="250px"></asp:TextBox><br />
            <br />
            <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
             Text="Submit to custom event log" /><br />
            <br />
            <asp:Label ID="Label1" runat="server"></asp:Label>&nbsp;</div>
        </form>
    </body>
    </html>
```

C#

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Diagnostics" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!EventLog.SourceExists("My Application"))
        {
            EventLog.CreateEventSource("My Application", "Application");
        }
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        EventLog el = new EventLog();
        el.Source = "My Application";
        el.WriteEntry(TextBox1.Text);

        Label1.Text = "ENTERED: " + TextBox1.Text;
    }
</script>
```

Again, for this to work, you must import the `System.Diagnostics` namespace. In the `Page_Load` event of the page, ASP.NET is checking whether the event source exists for `My Application`. If no such source exists in the Application event log, it is created using the `CreateEventSource()` method.

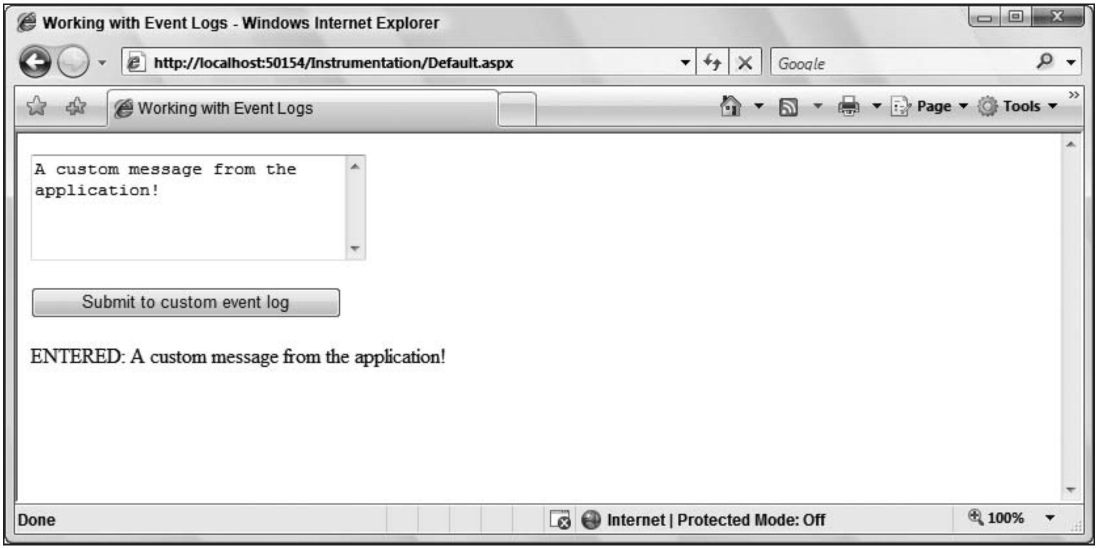
```
el.CreateEventSource("My Application", "Application")
```

The first parameter of the method takes the name of the source that you are creating. The second parameter of this method call takes the name of the event log that you are targeting. After this source has been created, you can start working with the `EventLog` object to place an entry into the system. First, the `EventLog` object is assigned a source. In this case, it is the newly created `My Application`. Using the `WriteEntry()` method, you can write to the specified event log. You can also assign the source and the message within the `WriteEntry()` method in the following manner:

```
el.WriteEntry("My Application", TextBox1.Text);
```

The ASP.NET page produces something similar to what is illustrated in Figure 32-3.

After this is done, you can look in the Event Viewer and see your entry listed in the Application event log. Figure 32-4 illustrates what happens when you double-click the entry.

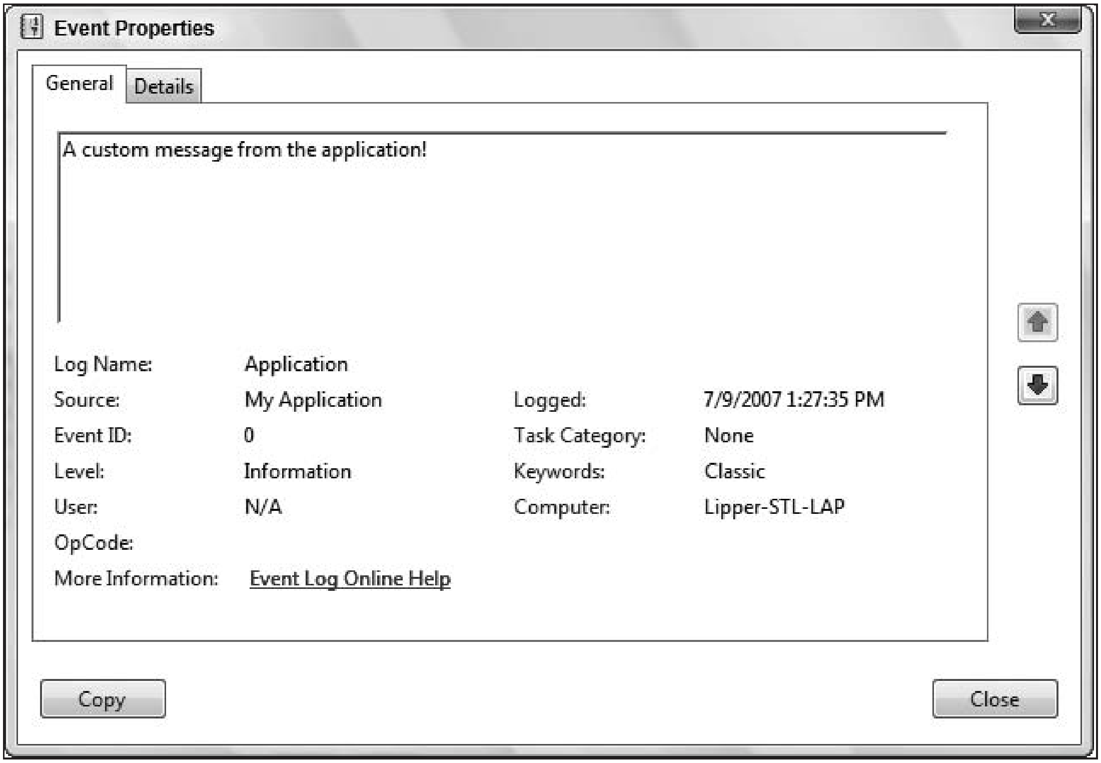

Working with Event Logs - Windows Internet Explorer
http://localhost:50154/Instrumentation/Default.aspx
Google
Working with Event Logs
Page
Tools
Done
Internet | Protected Mode: Off
100%

A custom message from the application!

Submit to custom event log

ENTERED: A custom message from the application!

Figure 32-3


Event Properties
General
Details
Copy
Close

A custom message from the application!

Log Name:	Application		
Source:	My Application	Logged:	7/9/2007 1:27:35 PM
Event ID:	0	Task Category:	None
Level:	Information	Keywords:	Classic
User:	N/A	Computer:	Lipper-STL-LAP
OpCode:			
More Information:	Event Log Online Help		

Figure 32-4

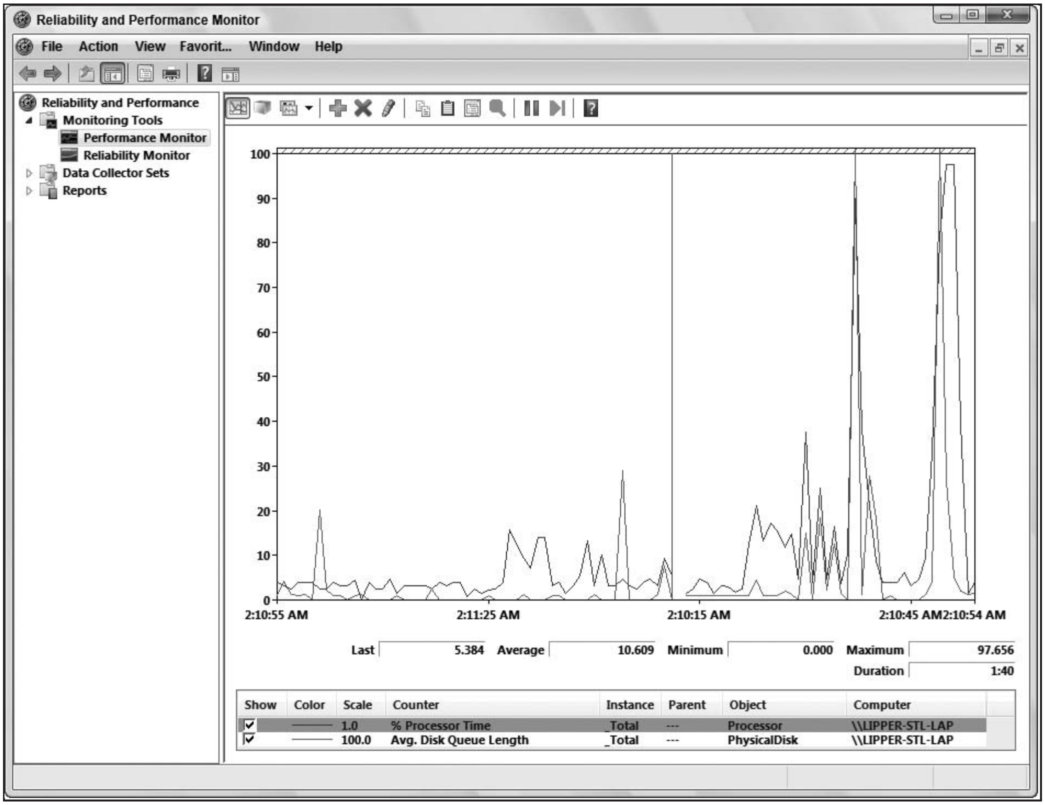
Later in this chapter, you see some of the automatic ways in which ASP.NET can record events for you in the event log and in some other data stores (such as Microsoft's SQL Server). Next, it is time to turn your attention to working with performance counters.

Using Performance Counters

Utilizing performance counters is important if you want to monitor your applications as they run. What exactly is monitored is up to you. A plethora of available performance counters are at your disposal in Windows and you will find that there are more than 60 counters specific to ASP.NET.

Viewing Performance Counters Through an Administration Tool

You can see these performance counters by opening the Performance dialog found in the Control Panel and then Administration Tools if you are using Windows XP. If you are using Windows Vista, select Control Panel ➪ System and Maintenance ➪ Performance Information and Tools ➪ Advanced Tools ➪ Open Reliability and Performance Monitor. Figure 32-5 shows the dialog opened in Windows Vista.


Reliability and Performance Monitor
File
Action
View
Favorit...
Window
Help
Reliability and Performance
Monitoring Tools
Performance Monitor
Reliability Monitor
Data Collector Sets
Reports
100
90
80
70
60
50
40
30
20
10
0
2:10:55 AM
2:11:25 AM
2:10:15 AM
2:10:45 AM
2:10:54 AM

Last 5.384 Average 10.609 Minimum 0.000 Maximum 97.656

Duration 1:40

Show	Color	Scale	Counter	Instance	Parent	Object	Computer
☑	———	1.0	% Processor Time	_Total	---	Processor	\\LIPPER-STL-LAP
☑	———	100.0	Avg. Disk Queue Length	_Total	---	PhysicalDisk	\\LIPPER-STL-LAP

Figure 32-5

Clicking the plus sign in the menu enables you to add more performance counters to the list. You will find a number of ASP.NET–specific counters in the list illustrated in Figure 32-6.

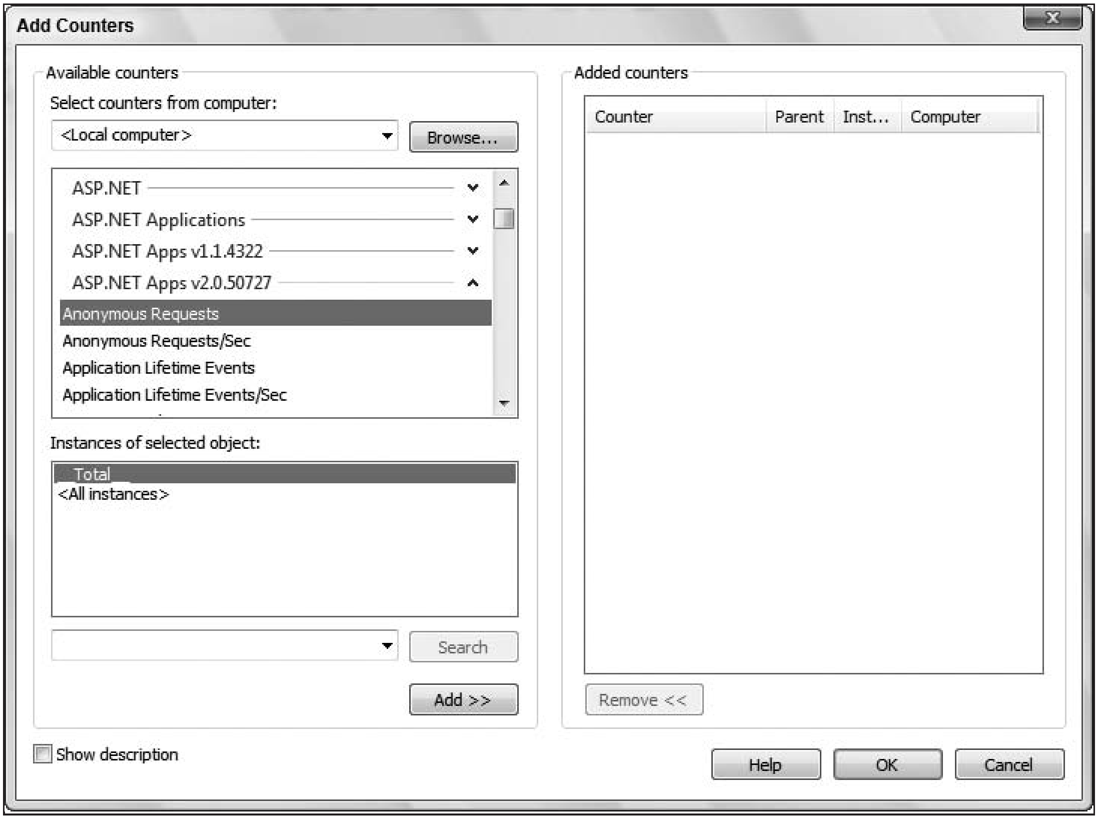

Add Counters
Available counters
Select counters from computer:
<Local computer>
Browse...
ASP.NET
ASP.NET Applications
ASP.NET Apps v1.1.4322
ASP.NET Apps v2.0.50727
Anonymous Requests
Anonymous Requests/Sec
Application Lifetime Events
Application Lifetime Events/Sec
Instances of selected object:
Total
<All instances>
Search
Add >>
Added counters
Counter
Parent
Inst...
Computer
Remove <<
☐ Show description
Help
OK
Cancel

Figure 32-6

The following list details some of the ASP.NET–specific performance counters that are at your disposal along with a definition of the counter (also available by checking the Show Description check box in Vista from within the dialog).

- **Application Restarts:** Number of times the application has been restarted during the Web server's lifetime.
- **Applications Running:** Number of currently running Web applications.
- **Audit Failure Events Raised:** Number of audit failures in the application since it was started.
- **Audit Success Events Raised:** Number of audit successes in the application since it was started.
- **Error Events Raised:** Number of error events raised since the application was started.
- **Infrastructure Error Events Raised:** Number of HTTP error events raised since the application was started.

- **Request Error Events Raised:** Number of runtime error events raised since the application was started.
- **Request Execution Time:** The number of milliseconds it took to execute the most recent request.
- **Request Wait Time:** The number of milliseconds the most recent request was waiting in the queue.
- **Requests Current:** The current number of requests, including those that are queued, currently executing, or waiting to be written to the client. Under the ASP.NET process model, when this counter exceeds the `requestQueueLimit` defined in the `processModel` configuration section, ASP.NET begins rejecting requests.
- **Requests Disconnected:** The number of requests disconnected because of communication errors or user terminations.
- **Requests Queued:** The number of requests waiting to be processed.
- **Requests Rejected:** The number of requests rejected because the request queue was full.
- **State Server Sessions Abandoned:** The number of sessions that have been explicitly abandoned.
- **State Server Sessions Active:** The current number of sessions currently active.
- **State Server Sessions Timed Out:** The number of sessions timed out.
- **State Server Sessions Total:** The number of sessions total.
- **Worker Process Restarts:** Number of times a worker process has restarted on the machine.
- **Worker Processes Running:** Number of worker processes running on the machine.

These are the performance counters for just the ASP.NET v2.0.50727 category. Here you will find categories for other ASP.NET–specific items such as:

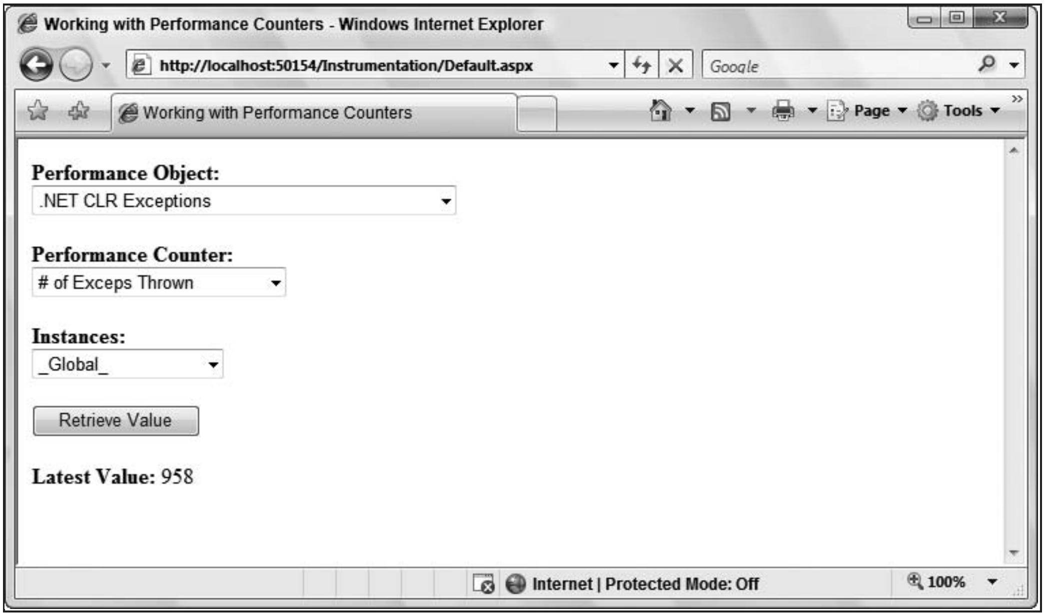
- ASP.NET
- ASP.NET Applications
- ASP.NET Apps v1.0.3705.288
- ASP.NET Apps v2.0.50727
- ASP.NET State Service
- ASP.NET v1.0.3705.288
- ASP.NET v2.0.50727

Performance counters can give you a pretty outstanding view of what is happening in your application. The data retrieved by a specific counter is not a continuous thing because the counter is really taking a snapshot of the specified counter every 400 milliseconds, so be sure to take that into account when analyzing the data produced.

Building a Browser-Based Administrative Tool

In addition to viewing the performance counters available to you through the Performance dialog, you can also get at the performance counter values programmatically. This is possible by working with the `System.Diagnostics` namespace in the .NET Framework. This namespace gives you access to performance-counter-specific objects such as the `PerformanceCounterCategory` and `PerformanceCounter` objects.

To show you how to work with these objects, this next example creates an ASP.NET page that enables you to view any value from a performance counter directly in the browser. To accomplish this task, create a basic ASP.NET page that includes three DropDownList controls, a Button control, and a Label control. This gives you the results presented in Figure 32-7.


Working with Performance Counters - Windows Internet Explorer
http://localhost:50154/Instrumentation/Default.aspx
Google
Working with Performance Counters
Page
Tools
Retrieve Value
Internet | Protected Mode: Off
100%

Performance Object: .NET CLR Exceptions

Performance Counter: # of Exceps Thrown

Instances: _Global_

Latest Value: 958

Figure 32-7

Listing 32-3 shows the code required for this figure.

Listing 32-3: Working with performance counters in ASP.NET

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Diagnostics" %>
<%@ Import Namespace="System.Collections.Generic" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        If Not Page.IsPostBack Then
            Dim pcc As List(Of String) = New List(Of String)

            For Each item As PerformanceCounterCategory In _
              PerformanceCounterCategory.GetCategories()
                pcc.Add(item.CategoryName)
            Next
```

Continued

```
        pcc.Sort()
        pcc.Remove(".NET CLR Data")

        DropDownList1.DataSource = pcc
        DropDownList1.DataBind()

        Dim myPcc As PerformanceCounterCategory
        myPcc = New PerformanceCounterCategory(DropDownList1.SelectedItem.Text)

        DisplayCounters(myPcc)
    End If
End Sub

Protected Sub DisplayCounters(ByVal pcc As PerformanceCounterCategory)
    DisplayInstances(pcc)

    Dim myPcc As List(Of String) = New List(Of String)

    If DropDownList3.Items.Count > 0 Then
        For Each pc As PerformanceCounter In _
          pcc.GetCounters(DropDownList3.Items(0).Value)
            myPcc.Add(pc.CounterName)
        Next
    Else
        For Each pc As PerformanceCounter In pcc.GetCounters()
            myPcc.Add(pc.CounterName)
        Next
    End If

    myPcc.Sort()

    DropDownList2.DataSource = myPcc
    DropDownList2.DataBind()
End Sub

Protected Sub DisplayInstances(ByVal pcc As PerformanceCounterCategory)
    Dim listPcc As List(Of String) = New List(Of String)

    For Each item As String In pcc.GetInstanceNames()
        listPcc.Add(item.ToString())
    Next

    listPcc.Sort()

    DropDownList3.DataSource = listPcc
    DropDownList3.DataBind()
End Sub

Protected Sub DropDownList1_SelectedIndexChanged(ByVal sender As Object, _
  ByVal e As System.EventArgs)
    Dim pcc As PerformanceCounterCategory
    pcc = New PerformanceCounterCategory(DropDownList1.SelectedItem.Text)
```

Continued

```
        DropDownList2.Items.Clear()
        DropDownList3.Items.Clear()

        DisplayCounters(pcc)
    End Sub

    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        Dim pc As PerformanceCounter

        If DropDownList3.Items.Count > 0 Then
            pc = New PerformanceCounter(DropDownList1.SelectedItem.Text, _
              DropDownList2.SelectedItem.Text, DropDownList3.SelectedItem.Text)
        Else
            pc = New PerformanceCounter(DropDownList1.SelectedItem.Text, _
              DropDownList2.SelectedItem.Text)
        End If

        Label1.Text = "<b>Latest Value:</b> " & pc.NextValue().ToString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Working with Performance Counters</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <strong>Performance Object:</strong><br />
        <asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True"
         OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
        </asp:DropDownList><br />
        <br />
        <strong>Performance Counter:</strong><br />
        <asp:DropDownList ID="DropDownList2" runat="server">
        </asp:DropDownList><br />
        <br />
        <strong>Instances:</strong><br />
        <asp:DropDownList ID="DropDownList3" runat="server">
        </asp:DropDownList><br />
        <br />
        <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
         Text="Retrieve Value" /><br />
        <br />
        <asp:Label ID="Label1" runat="server"></asp:Label></div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Diagnostics" %>
<%@ Import Namespace="System.Collections.Generic" %>
```

Continued

```
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            List<string> pcc = new List<string>();

            foreach (PerformanceCounterCategory item in
               PerformanceCounterCategory.GetCategories())
            {
                pcc.Add(item.CategoryName);
            }

            pcc.Sort();
            pcc.Remove(".NET CLR Data");

            DropDownList1.DataSource = pcc;
            DropDownList1.DataBind();

            PerformanceCounterCategory myPcc;
            myPcc = new
               PerformanceCounterCategory(DropDownList1.SelectedItem.Text);

            DisplayCounters(myPcc);
        }
    }

    void DisplayCounters(PerformanceCounterCategory pcc)
    {
        DisplayInstances(pcc);

        List<string> myPcc = new List<string>();

        if (DropDownList3.Items.Count > 0)
        {
            foreach (PerformanceCounter pc in
               pcc.GetCounters(DropDownList3.Items[0].Value))
            {
                myPcc.Add(pc.CounterName);
            }
        }
        else
        {
            foreach (PerformanceCounter pc in pcc.GetCounters())
            {
                myPcc.Add(pc.CounterName);
            }
        }

        myPcc.Sort();

        DropDownList2.DataSource = myPcc;
        DropDownList2.DataBind();
    }
```

Continued

```
        void DisplayInstances(PerformanceCounterCategory pcc)
        {
            List<string> listPcc = new List<string>();

            foreach (string item in pcc.GetInstanceNames())
            {
                listPcc.Add(item.ToString());
            }

            listPcc.Sort();

            DropDownList3.DataSource = listPcc;
            DropDownList3.DataBind();
        }

        protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
        {
            PerformanceCounterCategory pcc;
            pcc = new PerformanceCounterCategory(DropDownList1.SelectedItem.Text);

            DropDownList2.Items.Clear();
            DropDownList3.Items.Clear();

            DisplayCounters(pcc);
        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            PerformanceCounter pc;
            if (DropDownList3.Items.Count > 0)
            {
                pc = new PerformanceCounter(DropDownList1.SelectedItem.Text,
                  DropDownList2.SelectedItem.Text, DropDownList3.SelectedItem.Text);
            }
            else
            {
                pc = new PerformanceCounter(DropDownList1.SelectedItem.Text,
                  DropDownList2.SelectedItem.Text);
            }

            Label1.Text = "<b>Latest Value:</b> " + pc.NextValue().ToString();
        }
    </script>
```

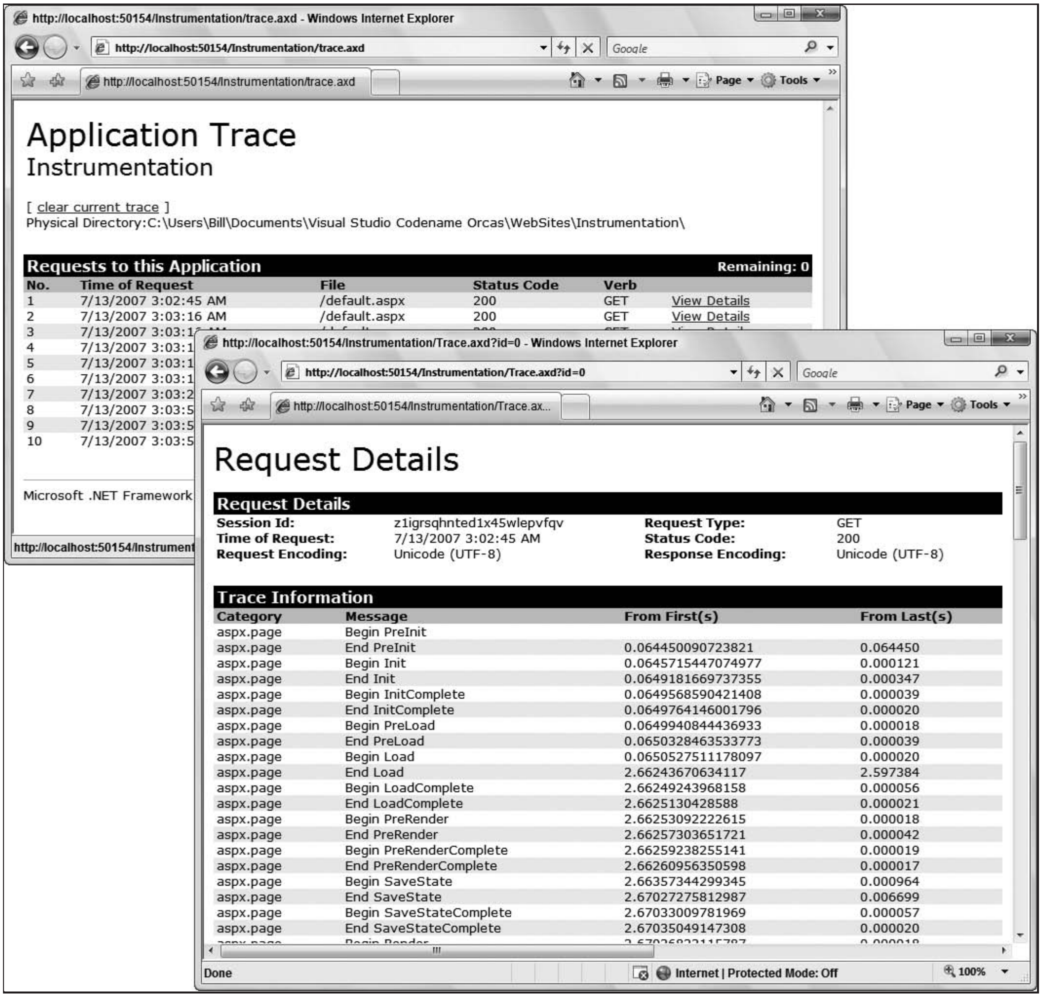
To make this work, you have to deal with only a couple of performance-counter objects such as the `PerformanceCounterCategory` and the `PerformanceCounter` objects. The first drop-down list is populated with all the categories available. These values are first placed in a `List(Of String)` object that enables you to call a `Sort()` method and also allows you to remove any categories you aren't interested in by using the `Remove()` method before binding to the DropDown list control.

The category selected in the first drop-down list drives what is displayed in the second and third drop-down lists. The second drop-down list displays a list of counters that are available for a particular category. You might tend to think that the third drop-down list of instances is based upon the counter that is selected, but instances are set at the category level.

When the button on the page is clicked, a new instance of the `PerformanceCounter` object is created and, as you can see from the example, it can be instantiated in several ways. The first constructor takes just a category name and a counter name, whereas the second constructor takes these two items plus the instance name utilized. After a `PerformanceCounter` is created, the `NextValue()` method pulls a sample from the specified item, thus producing the results that are illustrated in Figure 32-7.

Application Tracing

ASP.NET does an excellent job of displaying trace information either directly on the requested pages of your ASP.NET application or in a trace log that can be found at `http://[server]/[application]/trace.axd`. Sample screenshots of both of these scenarios appear in Figure 32-8.


http://localhost:50154/Instrumentation/trace.axd - Windows Internet Explorer
http://localhost:50154/Instrumentation/trace.axd
Google
http://localhost:50154/Instrumentation/trace.axd
Page
Tools
http://localhost:50154/Instrument
http://localhost:50154/Instrumentation/Trace.axd?id=0 - Windows Internet Explorer
http://localhost:50154/Instrumentation/Trace.axd?id=0
Google
http://localhost:50154/Instrumentation/Trace.ax...
Page
Tools
Done
Internet | Protected Mode: Off
100%

Application Trace

Instrumentation

[clear current trace]
Physical Directory:C:\Users\Bill\Documents\Visual Studio Codename Orcas\WebSites\Instrumentation\

Requests to this Application **Remaining: 0**

No.	Time of Request	File	Status Code	Verb	
1	7/13/2007 3:02:45 AM	/default.aspx	200	GET	View Details
2	7/13/2007 3:03:16 AM	/default.aspx	200	GET	View Details
3	7/13/2007 3:03:1				
4	7/13/2007 3:03:1				
5	7/13/2007 3:03:1				
6	7/13/2007 3:03:1				
7	7/13/2007 3:03:2				
8	7/13/2007 3:03:5				
9	7/13/2007 3:03:5				
10	7/13/2007 3:03:5				

Microsoft .NET Framework

Request Details

Request Details

Session Id:	z1igrsqhnted1x45wlepvfqv	**Request Type:**	GET
Time of Request:	7/13/2007 3:02:45 AM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0.064450090723821	0.064450
aspx.page	Begin Init	0.0645715447074977	0.000121
aspx.page	End Init	0.0649181669737355	0.000347
aspx.page	Begin InitComplete	0.0649568590421408	0.000039
aspx.page	End InitComplete	0.0649764146001796	0.000020
aspx.page	Begin PreLoad	0.0649940844436933	0.000018
aspx.page	End PreLoad	0.0650328463533773	0.000039
aspx.page	Begin Load	0.0650527511178097	0.000020
aspx.page	End Load	2.66243670634117	2.597384
aspx.page	Begin LoadComplete	2.66249243968158	0.000056
aspx.page	End LoadComplete	2.6625130428588	0.000021
aspx.page	Begin PreRender	2.66253092222615	0.000018
aspx.page	End PreRender	2.66257303651721	0.000042
aspx.page	Begin PreRenderComplete	2.66259238255141	0.000019
aspx.page	End PreRenderComplete	2.66260956350598	0.000017
aspx.page	Begin SaveState	2.66357344299345	0.000964
aspx.page	End SaveState	2.67027275812987	0.006699
aspx.page	Begin SaveStateComplete	2.67033009781969	0.000057
aspx.page	End SaveStateComplete	2.67035049147308	0.000020

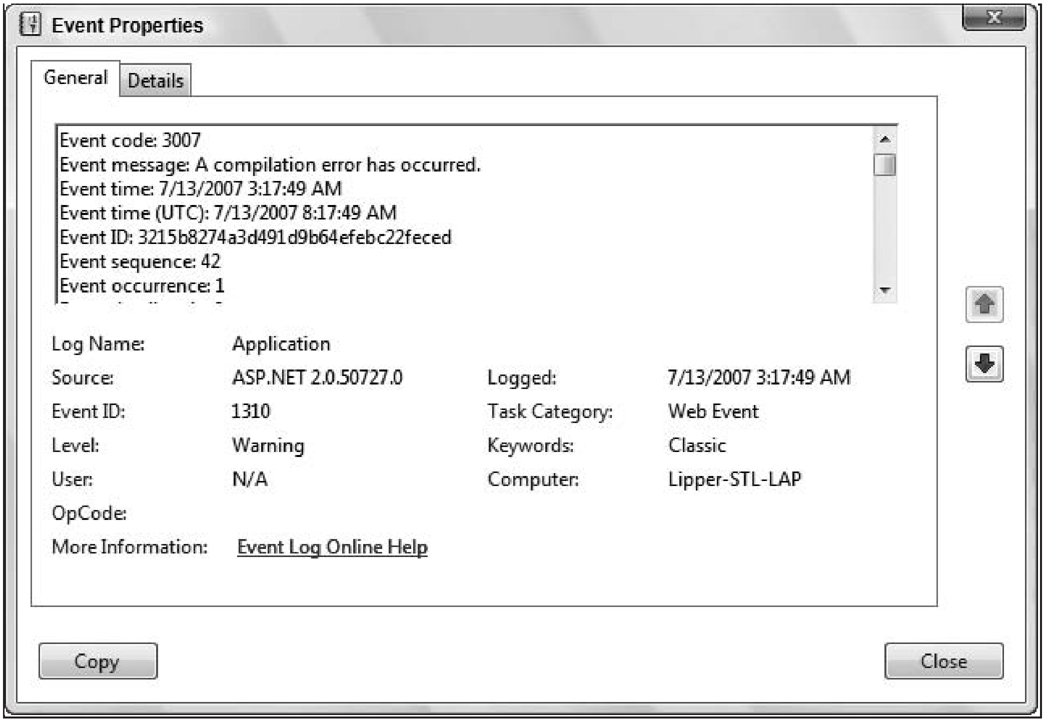
Figure 32-8

You can get a lot of detailed information on working with tracing for instrumentation in Chapter 24.

Understanding Health Monitoring

One of the more exciting instrumentation capabilities provided by ASP.NET is the health monitoring system introduced with ASP.NET 2.0. ASP.NET health monitoring is built around various health monitoring events (which are referred to as *Web events*) occurring in your application. Using the health monitoring system enables you to use the event logging for Web events such as failed logins, application starts and stops, or any unhandled exceptions. The event logging can occur in more than one place; therefore, you can log to the event log or even back to a database. In addition to this disk-based logging, you can also use the system to e-mail health monitoring information.

By default, the health monitoring system is already enabled. All default errors and failure audits are logged into the event logs on your behalf. For instance, throwing an error produces an entry in the event log, as illustrated in Figure 32-9.



```
Event code: 3007
Event message: A compilation error has occurred.
Event time: 7/13/2007 3:17:49 AM
Event time (UTC): 7/13/2007 8:17:49 AM
Event ID: 3215b8274a3d491d9b64efebc22feced
Event sequence: 42
Event occurrence: 1
```

Log Name:	Application		
Source:	ASP.NET 2.0.50727.0	Logged:	7/13/2007 3:17:49 AM
Event ID:	1310	Task Category:	Web Event
Level:	Warning	Keywords:	Classic
User:	N/A	Computer:	Lipper-STL-LAP
OpCode:			
More Information:	Event Log Online Help		

Figure 32-9

By default, these errors are registered in the event logs, but you can also record these events in a couple of other places. You define where you record these event messages through the various providers available to the health monitoring system. These providers are briefly covered next.

The Health Monitoring Provider Model

Quite a bit of information about what a provider model is and how the health monitoring system works with providers is covered in Chapter 12, but a short review of the providers available to the health monitoring system is warranted here as well.

The health monitoring system has the most built-in providers in ASP.NET. A diagram of the available providers is shown in Figure 32-10.

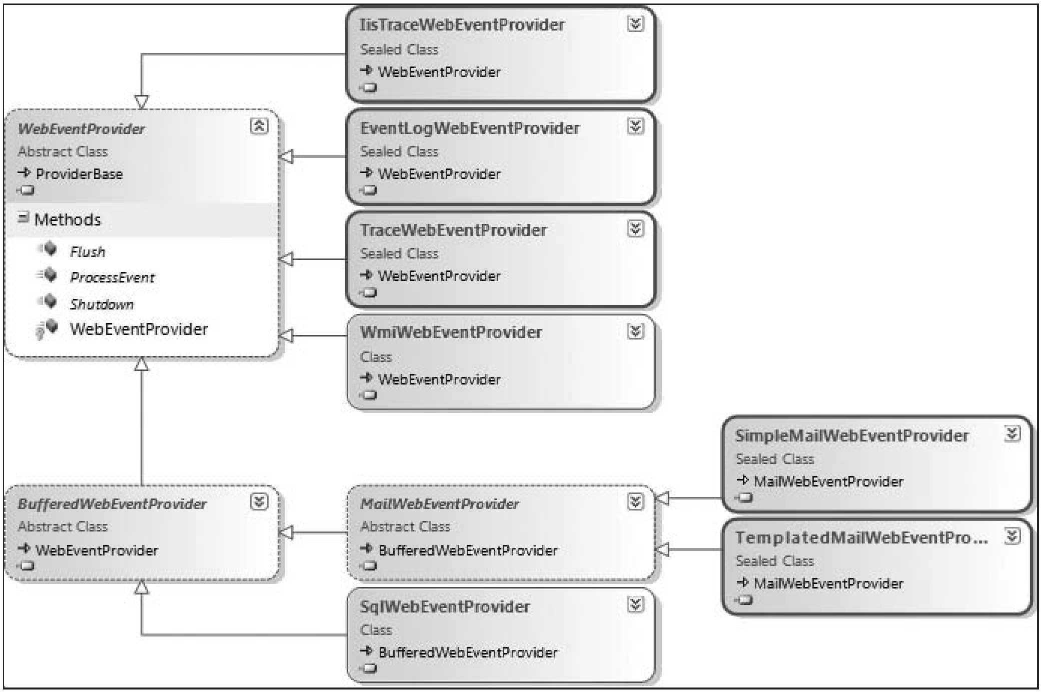

IisTraceWebEventProvider
Sealed Class
WebEventProvider
WebEventProvider
Abstract Class
ProviderBase
Methods
Flush
ProcessEvent
Shutdown
WebEventProvider
EventLogWebEventProvider
Sealed Class
WebEventProvider
TraceWebEventProvider
Sealed Class
WebEventProvider
WmiWebEventProvider
Class
WebEventProvider
SimpleMailWebEventProvider
Sealed Class
MailWebEventProvider
BufferedWebEventProvider
Abstract Class
WebEventProvider
MailWebEventProvider
Abstract Class
BufferedWebEventProvider
TemplatedMailWebEventPro...
Sealed Class
MailWebEventProvider
SqlWebEventProvider
Class
BufferedWebEventProvider

Figure 32-10

Seven providers are available to the health monitoring system right out of the box:

- `System.Web.Management.EventLogWebEventProvider`: Enables you to use the ASP.NET health monitoring system to record security operation errors and all other errors into the Windows event log.
- `System.Web.Management.SimpleMailWebEventProvider`: Allows you to use the ASP.NET health monitoring system to send error information in an e-mail.
- `System.Web.Management.TemplatedMailWebEventProvider`: Similar to the `SimpleMailWebEventProvider`, the `TemplatedMailWebEventProvider` class this provider lets you send error information in a templated e-mail. Templates are defined using a standard `.aspx` page.
- `System.Web.Management.SqlWebEventProvider`: Enables you to use the ASP.NET health monitoring system to store error information in SQL Server. Like the other SQL providers for the other systems in ASP.NET, the `SqlWebEventProvider` stores error information in SQL Server Express Edition by default.
- `System.Web.Management.TraceWebEventProvider`: Enables you to use the ASP.NET health monitoring system to send error information to the ASP.NET page tracing system.
- `System.Web.Management.IisTraceWebEventProvider`–: Provides you with the capability to use the ASP.NET health monitoring system to send error information to the IIS tracing system.

- `System.Web.Management.WmiWebEventProvider`: Enables you to connect the ASP.NET health monitoring system to the Windows Management Instrumentation (WMI) event provider.

Health Monitoring Configuration

By default, the `EventLogWebEventProvider` object is what is utilized for all errors and failure audits. These rules are defined along with the providers and what Web events to trap in the root `web.config` file found at `C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG`. All the information for the health monitoring section is defined within the `<healthMonitoring>` section of this file. Even though these items are defined in the root `web.config` file, you can also override these settings and define them yourself in your application's `web.config` file.

To define the health monitoring capabilities and behaviors in your `web.config` file, place a `<healthMonitoring>` section within the `< system.web >` section of your configuration file:

```
<configuration>
   <system.web>

    <healthMonitoring enabled="true">

    </healthMonitoring>

   </system.web>
</configuration>
```

The `<healthMonitoring>` section can include a number of subsections, including the following:

- `<bufferModes>`
- `<eventMappings>`
- `<profiles>`
- `<providers>`
- `<rules>`

You next look at some of the core sections and how you can use these sections to define health monitoring tasks in your application's `web.config` file.

<eventMappings>

The `<eventMappings>` section of the health monitoring system allows you to define friendly names for specific events that can be captured. You can declare a number of events in this section, but doing so doesn't mean that they are utilized automatically. Remember that when placing events in this section, you are just simply defining them in this configuration file for reuse in other sections. Listing 32-4 shows an example of event mapping.

Listing 32-4: Using the <eventMappings> section

```
<configuration>
   <system.web>
```

Continued

```
            <healthMonitoring enabled="true">

               <eventMappings>

                  <clear />

                  <add name="All Events"
                   type="System.Web.Management.WebBaseEvent,System.Web,
                     Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a"
                   startEventCode="0" endEventCode="2147483647" />
                  <add name="Heartbeats"
                   type="System.Web.Management.WebHeartbeatEvent,System.Web,
                     Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a"
                   startEventCode="0" endEventCode="2147483647" />
                  <add name="Application Lifetime Events"
                   type="System.Web.Management.WebApplicationLifetimeEvent,System.Web,
                     Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a"
                   startEventCode="0" endEventCode="2147483647" />
                  <add name="Request Processing Events"
                   type="System.Web.Management.WebRequestEvent,System.Web,
                     Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a"
                   startEventCode="0" endEventCode="2147483647" />
                  <add name="All Errors"
                   type="System.Web.Management.WebBaseErrorEvent,System.Web,
                     Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a"
                   startEventCode="0" endEventCode="2147483647" />
                  <add name="Infrastructure Errors"
                   type="System.Web.Management.WebErrorEvent,System.Web,
                     Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a"
                   startEventCode="0" endEventCode="2147483647" />
                  <add name="Request Processing Errors"
                   type="System.Web.Management.WebRequestErrorEvent,System.Web,
                     Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a"
                   startEventCode="0" endEventCode="2147483647" />
                  <add name="All Audits"
                   type="System.Web.Management.WebAuditEvent,System.Web,
                     Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a"
                   startEventCode="0" endEventCode="2147483647" />
                  <add name="Failure Audits"
                   type="System.Web.Management.WebFailureAuditEvent,System.Web,
                     Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a"
                   startEventCode="0" endEventCode="2147483647" />
                  <add name="Success Audits"
                   type="System.Web.Management.WebSuccessAuditEvent,System.Web,
                     Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a"
                   startEventCode="0" endEventCode="2147483647" />
               </eventMappings>

            </healthMonitoring>

         </system.web>
      </configuration>
```

From this section, you can see that a number of different event types are defined within this `<eventMappings>` section. Not all these definitions are required, just the ones you are interested in working with. Because these definitions are important, look at how the first one is listed:

```
<add name="All Events"
 type="System.Web.Management.WebBaseEvent,System.Web,
   Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a"
 startEventCode="0" endEventCode="2147483647" />
```

The `<add>` element takes a number of different attributes. The first, `name`, allows you to give a user-friendly name that can be used later in one of the other sections of the document. In this case, the friendly name provided is `All Events`.

The next attribute is the `type` attribute. This enables you to define the .NET event object with which this event mapping is associated. In this case, it is the base event used by all the other event types as a base class. Although the object definition with the `type` attribute is shown on multiple lines in this book, you should define it on a single line in order for it to work. All the possible Web events it might associate with are found in the `System.Web.Management` namespace.

Each Web Event that is recorded has a code associated with it, and you can use the `startEventCode` attribute to define a starting point for this code definition. In the preceding example, the event code starts at `0` and increments from there until it reaches the number defined in the `endEventCode` attribute. In this case, the ending event code is `2147483647`.

The following table defines each event type you can find in the `System.Web.Management` namespace.

Web Event (System.Web.Management)	**Description**
`WebBaseEvent`	The `WebBaseEvent` class is the base class for all Web event types. You can, therefore, use this instance within the `<eventMappings>` section in order to create a definition to capture all events.
`WebHeartbeatEvent`	This event defines Web events that are recorded only at specific intervals instead of every time they occur.
`WebApplicationLifetimeEvent`	This event defines Web events that occur on the scale of the application and its lifecycle. These types of events include application starts and stops as well as compilation starts and stops.
`WebRequestEvent`	This event defines Web events that occur during a request cycle and include items such as when a transaction for a request is committed or aborted.
`WebBaseErrorEvent`	The `WebBaseErrorEvent` class is the base class for all Web events that deal with error types. You can, therefore, use this instance within the `<eventMappings>` section to create a definition to capture all error events.
`WebErrorEvent`	This event defines Web events that occur because of configuration errors or any compilation or parsing errors.

Web Event (System.Web.Management)	**Description**
`WebRequestErrorEvent`	This event defines Web events that occur because requests are aborted or requests are larger than allowable as defined by the `maxRequestLength` attribute in the configuration file. It also includes any validation errors or any unhandled exceptions.
`WebAuditEvent`	The `WebAuditEvent` class is the base class for all Web events that deal with login attempts. These attempts can either fail or succeed. You can use this instance within the `<eventMappings>` section to write a definition to capture all login attempts.
`WebFailureAuditEvent`	This event defines Web events that occur because of failed login attempts.

Now that all the error definitions are in place within the `<eventMappings>` section, the next step is to further define the structure your health monitoring system will take by detailing the `<providers>` section, which is also found within the `<healthMonitoring>` section.

<providers>

As you saw earlier in Figure 32-10, seven different providers that you can use within the health monitoring system are available to you out of the box. By default, ASP.NET records these events to the event logs using the `EventLogWebEventProvider`, but you can also modify the provider model so that it uses any of the other providers available. You can also build your own custom providers so you can record these Web events to any data store you want.

You can find more information on building custom providers in Chapter 13.

Providers are declared within the `<providers>` section, which is nested within the `<healthMonitoring>` section of the document. Within the root `web.config` file found at `C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG`, you find a short list of declared providers. You can also declare other providers in your application's `web.config` file as presented in Listing 32-5.

Listing 32-5: Using the <providers> section

```
<configuration>
   <system.web>

      <healthMonitoring enabled="true">

         <eventMappings>
            <!-- Code removed for clarity -->
         </eventMappings>

         <providers>

            <clear />
```

```
                <add name="EventLogProvider"
                 type="System.Web.Management.EventLogWebEventProvider,System.Web,
                   Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a" />
                <add name="SqlWebEventProvider"
                 connectionStringName="LocalSqlServer"
                 maxEventDetailsLength="1073741823"
                 buffer="false" bufferMode="Notification"
                 type="System.Web.Management.SqlWebEventProvider,System.Web,
                   Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a" />
                <add name="WmiWebEventProvider"
                 type="System.Web.Management.WmiWebEventProvider,System.Web,
                 Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a" />

            </providers>

        </healthMonitoring>

    </system.web>
</configuration>
```

In this example, you see three separate health monitoring providers declared in the `web.config` file. A provider is defined for storing Web events in the event logs, another for storing in SQL Server, and finally another for the Windows Management Instrumentation (WMI) system.

By declaring these providers within the `<providers>` section of the `web.config` file, you don't ensure that these providers are *actually* utilized. Instead, they are simply defined and ready for you to use. You specify which providers to use within the `<rules>` section of the `<healthMonitoring>` section, which is defined shortly.

You can add defined providers by using the `<add />` element within the `<providers>` section. Within the `<add />` element, you find a series of available attributes. The first one is the `name` attribute. This allows you to provide a friendly name to the defined provider instance that you will use later when specifying the provider to actually use. The name can be anything you want. The `type` attribute allows you to define the class used for the provider. In some cases, this is all you need (for example, for the event log provider). If you are working with a database-linked provider, however, you must further define things like the `connectionString` attribute along with the `buffer` and `bufferMode` attributes. These are covered in more detail later.

After your providers are defined in the `<healthMonitoring>` section of the document, the next step is to determine which Web events to work with and which provider(s) should be utilized in the monitoring of these events. Both these operations are accomplished from within the `<rules>` section.

<rules>

The `<rules>` section allows you to define the Web events to monitor and the providers to tie them to when one of the monitored Web events is actually triggered. When you are using the health monitoring system, you can actually assign multiple providers to watch for the same Web events. This means that you can store the same Web event in both the event logs *and* in SQL Server. That is pretty powerful. Listing 32-6 provides an example of using the `<rules>` section within your application's `web.config` file.

Listing 32-6: Using the <rules> section

```
<configuration>
   <system.web>

      <healthMonitoring enabled="true">

         <eventMappings>
            <!-- Code removed for clarity -->
         </eventMappings>

         <providers>
            <!-- Code removed for clarity -->
         </providers>

         <rules>

            <clear />

            <add name="All Errors Default" eventName="All Errors"
             provider="EventLogProvider"
             profile="Default" minInstances="1" maxLimit="Infinite"
             minInterval="00:01:00" custom="" />
            <add name="Failure Audits Default" eventName="Failure Audits"
             provider="EventLogProvider" profile="Default" minInstances="1"
             maxLimit="Infinite" minInterval="00:01:00" custom="" />

         </rules>

      </healthMonitoring>

   </system.web>
</configuration>
```

In this example, two types of Web events are being recorded. You specify rules (the Web events to monitor) by using the `<add />` element within the `<rules>` section. The `name` attribute allows you to provide a friendly name for the rule definition. The `eventName` is an important attribute because it takes a value of the Web event to monitor. These names are the friendly names that you defined earlier within the `<eventMappings>` section of the `<healthMonitoring>` section. The first `<add />` element provides a definition of monitoring for `All Errors` via the `eventName` attribute. Looking back, you can see that this was, indeed, defined in the `<eventMappings>` section.

```
<eventMappings>
   <add name="All Errors"
    type="System.Web.Management.WebBaseErrorEvent,System.Web,
       Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a"
    startEventCode="0" endEventCode="2147483647" />
</eventMappings>
```

After specifying which Web event to work with through the `eventName` attribute, the next step is to define which provider to use for this Web event. You do this using the `provider` attribute. In the case of the first `<add />` element, the `EventLogProvider` is utilized. Again, this is the friendly name that was used for the provider definition in the <providers> section, as shown here:

```
<providers>
   <add name="EventLogProvider"
    type="System.Web.Management.EventLogWebEventProvider,System.Web,
      Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a" />
</providers>
```

The three attributes discussed so far are required attributes. The rest are considered optional attributes. The `minInstances` attribute defines the minimum number of times this Web event must occur before it is logged. The `maxLimit` attribute defines the maximum number of instances of the defined Web event that can be recorded. If instances are likely to occur because of some continuous loop, you might want to add a value here. The `minInterval` attribute denotes the minimum time allowed between log entries. This means that if the `minInterval` is set to `00:01:00` and two Web events being monitored occur within 15 seconds of each other, the first one is recorded but the second instance is discarded because it falls within the 1-minute setting.

Within the `<rules>` section, you can also see a `profile` attribute defined in the `<add />` elements of the rules defined. The value of this attribute comes from a definition that is supplied from the `<profiles>` section of the `<healthMonitoring>` section. This section is discussed next.

<profiles>

The `<profiles>` section enables you to define some of the behaviors for Web event monitoring. These definitions are utilized by any number of rule definitions within the `<rules>` section. An example use of the `<profiles>` section is illustrated in Listing 32-7.

Listing 32-7: Using the <profiles> section

```
<configuration>
   <system.web>

      <healthMonitoring enabled="true">

         <eventMappings>
            <!-- Code removed for clarity -->
         </eventMappings>

         <providers>
            <!-- Code removed for clarity -->
         </providers>

         <rules>
            <!-- Code removed for clarity -->
         </rules>

         <profiles>

            <clear />

            <add name="Default" minInstances="1" maxLimit="Infinite"
             minInterval="00:01:00" custom="" />
```

Continued

```
            <add name="Critical" minInstances="1" maxLimit="Infinite"
             minInterval="00:00:00" custom="" />

         </profiles>

      </healthMonitoring>

   </system.web>
</configuration>
```

As with the other sections, you add a profile definition by using the `<add />` element within the `<profiles>` section. The `name` attribute allows you to provide a friendly name that is then utilized from the definitions placed within the `<rules>` section, as illustrated here:

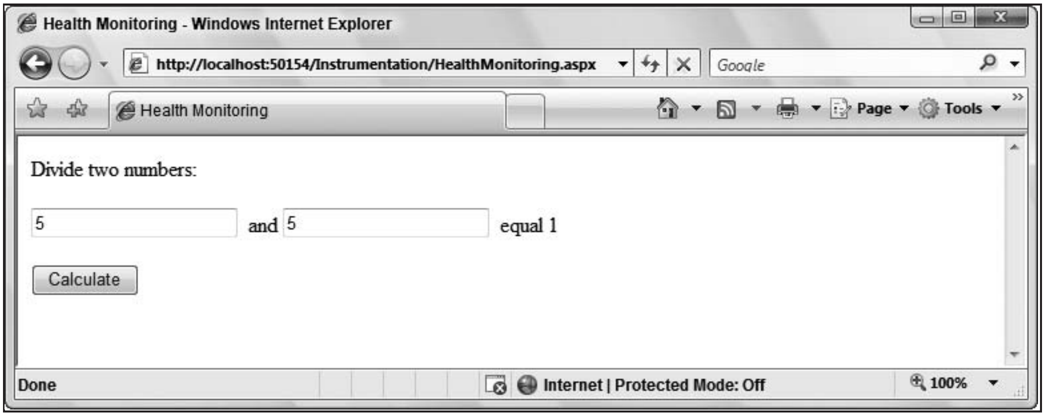
```
<rules>
   <add name="All Errors Default" eventName="All Errors"
    provider="EventLogProvider"
    profile="Default" minInstances="1" maxLimit="Infinite"
    minInterval="00:01:00" custom="" />
</rules>
```

The definitions in the `<profiles>` section also use the attributes `minInstances`, `maxLimit`, and `minInterval`. These have the same meanings as if they were used directly in the `<rules>` section (see the explanation in the previous section). The idea here, however, is that you can more centrally define these values and use them across multiple rule definitions.

Writing Events via Configuration: Running the Example

Using the sample `<healthMonitoring>` section (illustrated in the previous listings in this chapter), you can now write all errors as well as audits that fail (failed logins) to the event log automatically.

To test this construction in the `<healthMonitoring>` section, create a simple ASP.NET page that allows you to divide two numbers and then show the result of the division on the screen. An example ASP.NET page is presented in Figure 32-11.

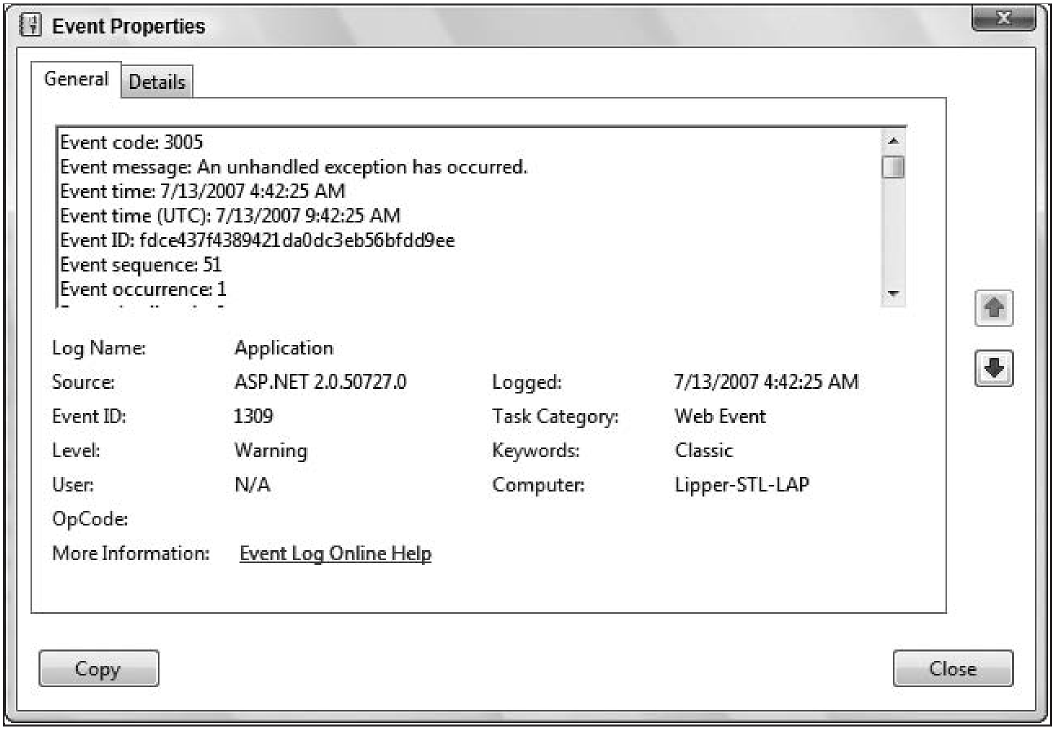

Health Monitoring - Windows Internet Explorer
http://localhost:50154/Instrumentation/HealthMonitoring.aspx
Google
Health Monitoring
Page
Tools
Calculate
Done
Internet | Protected Mode: Off
100%

Divide two numbers:

5 and 5 equal 1

Figure 32-11

As you can see from this page, you can enter a single number in the first text box and another in the second text box and press the calculate button. This allows the two numbers to be divided. The result appears in a Label control on the page. In this example, the code intentionally does not use any exception handling. Therefore, if the end user tried to divide 5 by 0 (zero), an exception is thrown. This event, in turn, causes the exception to be written to the event log. In fact, if you run a similar example and look in the event log, you find that the error has indeed been written as you have specified it should be within the `web.config` file. The report from the event log is presented in Figure 32-12.



```
Event code: 3005
Event message: An unhandled exception has occurred.
Event time: 7/13/2007 4:42:25 AM
Event time (UTC): 7/13/2007 9:42:25 AM
Event ID: fdce437f4389421da0dc3eb56bfdd9ee
Event sequence: 51
Event occurrence: 1
```

Log Name:	Application		
Source:	ASP.NET 2.0.50727.0	Logged:	7/13/2007 4:42:25 AM
Event ID:	1309	Task Category:	Web Event
Level:	Warning	Keywords:	Classic
User:	N/A	Computer:	Lipper-STL-LAP
OpCode:			
More Information:	Event Log Online Help		

Figure 32-12

Routing Events to SQL Server

Pushing Web events to the event log is something most Web applications do and is also something you should consider if you have overridden these built-in (and default) features. Even more powerful than writing them to the event log, however, is being able to write them to a database.

Writing them to a database allows you to actually create a larger history, makes them more secure, and allows you to more easily retrieve the values and use them in any type of administration application that you might create.

This capability was briefly introduced in Chapter 1 and is not that difficult to work with. If you work from the same `<healthMonitoring>` section created earlier in this chapter, writing the events to SQL Server is as simple as adding some `<add />` elements to the various sections.

As shown in the previous `<providers>` section, a declaration actually exists for recording events into SQL Server. It is presented here again:

```
<providers>

   <clear />

   <add name="EventLogProvider"
    type="System.Web.Management.EventLogWebEventProvider,System.Web,
      Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a" />

   <add name="SqlWebEventProvider"
    connectionStringName="LocalSqlServer"
    maxEventDetailsLength="1073741823"
    buffer="false" bufferMode="Notification"
    type="System.Web.Management.SqlWebEventProvider,System.Web,
      Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a" />

   <add name="WmiWebEventProvider"
    type="System.Web.Management.WmiWebEventProvider,System.Web,
      Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a" />

</providers>
```

If you look over this bit of code, you see the definition to record events into SQL Server is presented in the bold section within the `<providers>` section. The section in bold shows a connection to the SQL Server instance that is defined by the `LocalSqlServer` connection string name. You can find this connection string in the `machine.config` file, and you see that it points to an auto-generated SQL Server Express file that ASP.NET can work with.

```
<connectionStrings>
   <add name="LocalSqlServer" connectionString="data source=.\SQLEXPRESS;Integrated
    Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;User Instance=true"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

> *To work with Microsoft's SQL Server 2005, you simply create a new connection string within your ASP.NET application's* `web.config` *file and reference the friendly name provided to this connection from the* `<add />` *element of the* `<providers>` *section.*

Although this SQL Server instance is declared and defined, it is not utilized by any of the rules that you have established. Looking at the previous sections, you can see that within the `<rules>` section of the health monitoring section of the `web.config`, only two rules exist: a rule to write all errors to the event log and another rule to write failure audits to the event log. To add a new rule to write errors (all errors again) to SQL Server, you use the construction shown in Listing 32-8.

Listing 32-8: Creating a rule to write events to SQL Server

```
<configuration>
   <system.web>

      <healthMonitoring enabled="true">
```

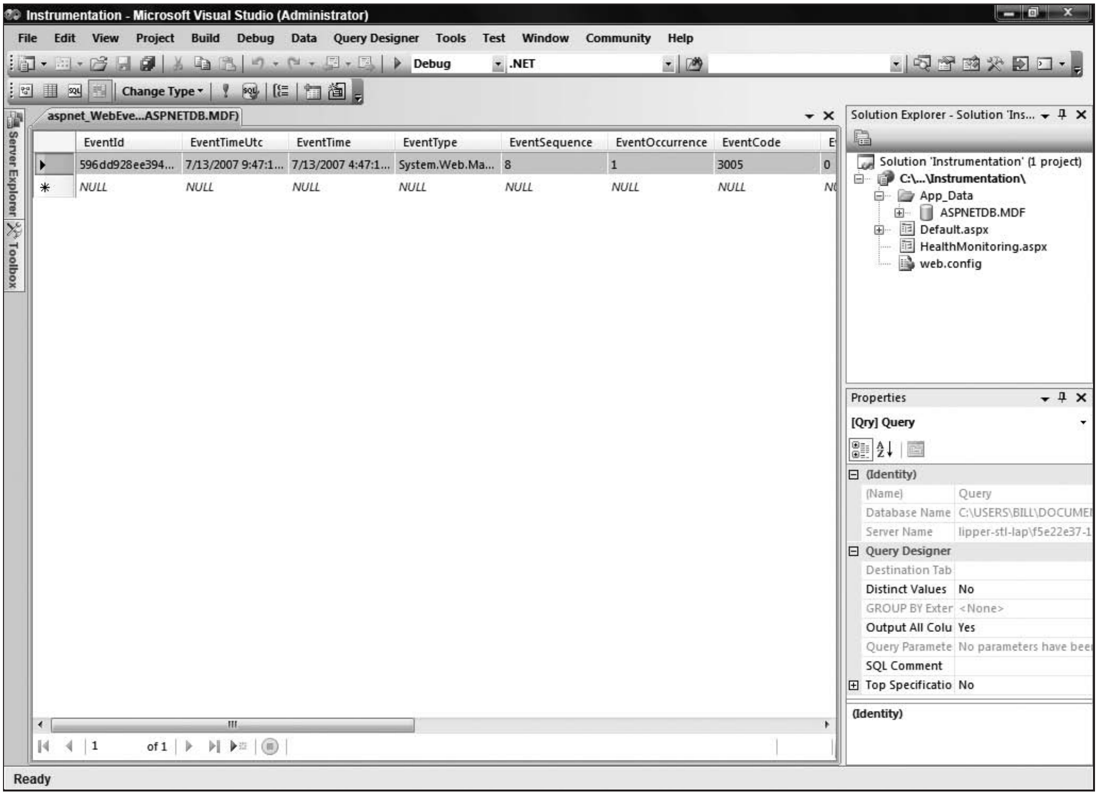
```
            <eventMappings>
               <!-- Code removed for clarity -->
            </eventMappings>

            <providers>
               <!-- Code removed for clarity -->
            </providers>

            <rules>

               <clear />

               <add name="All Errors Default" eventName="All Errors"
                provider="EventLogProvider"
                profile="Default" minInstances="1" maxLimit="Infinite"
                minInterval="00:01:00" custom="" />
               <add name="Failure Audits Default" eventName="Failure Audits"
                provider="EventLogProvider" profile="Default" minInstances="1"
                maxLimit="Infinite" minInterval="00:01:00" custom="" />
               <add name="All Errors SQL Server" eventName="All Errors"
                provider="SqlWebEventProvider"
                profile="Default" minInstances="1" maxLimit="Infinite"
                minInterval="00:01:00" custom="" />

            </rules>

         </healthMonitoring>

      </system.web>
   </configuration>
```

To be able to write to SQL Server, you must create a new `<add />` element within the `<rules>` section of the document. The friendly name provided via the `name` attribute must be unique in the list of defined rules or you encounter an error. In this case, the name of the rule is `All Errors SQL Server` and like the `All Errors Default` rule, it points to the event mapping of `All Errors` as shown via the `eventName` attribute. The `provider` attribute then points to the SQL Server definition contained within the `<providers>` section. In this case, it is `SqlWebEventProvider`.

With all this in place, run the ASP.NET page (shown earlier) that allows you to throw an error by dividing by zero. If you run this page a couple of times, you see that ASP.NET has automatically created an `ASPNETDB.MDF` file in your project (if you didn't already have one).

You can then open the SQL Server Express Edition database and expand the Tables node in the Server Explorer of Visual Studio. In this list of available tables, you find a new table called `aspnet_WebEvent_Events`. Right-click this table and select Show Table Data from the menu. This gives you something similar to what is illustrated in Figure 32-13.

As you can see from this figure, the entire event is now stored within SQL Server (it is the first and only event at the moment). One of the nice things about this example is that not only is this event recorded to the database, but it is also written to the event log because you can use more than a single provider for each event. You can just as easily use a provider that comes from `MailWebEventProvider` and send the error out as an e-mail message.



Instrumentation - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Data Query Designer Tools Test Window Community Help
Debug
.NET
Change Type
aspnet_WebEve...ASPNETDB.MDF)
Server Explorer
Toolbox
Solution Explorer - Solution 'Ins...
Solution 'Instrumentation' (1 project)
C:\...\Instrumentation\
App_Data
ASPNETDB.MDF
Default.aspx
HealthMonitoring.aspx
web.config
Properties
[Qry] Query
1
of 1
Ready

	EventId	EventTimeUtc	EventTime	EventType	EventSequence	EventOccurrence	EventCode	E
▶	596dd928ee394...	7/13/2007 9:47:1...	7/13/2007 4:47:1...	System.Web.Ma...	8	1	3005	0
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	N

⊟ (Identity)	
(Name)	Query
Database Name	C:\USERS\BILL\DOCUME
Server Name	lipper-stl-lap\f5e22e37-1
⊟ Query Designer	
Destination Tab	
Distinct Values	No
GROUP BY Exter	<None>
Output All Colu	Yes
Query Paramete	No parameters have bee
SQL Comment	
⊞ Top Specificatio	No

(Identity)

Figure 32-13

Buffering Web Events

When you start working with Web events that you want to write to a database (such as SQL Server) or sent via an e-mail, you soon realize that doing so can really lock up your database or result in a huge amount of e-mail (especially if your application is sent into some perpetual loop of errors). For this reason, you can see why these providers inherit from `BufferedWebEventProvider`.

`BufferedWebEventProvider` does exactly what it says — it *buffers* (or queues) your collected Web events before performing the specified action upon them. The reason you might want to buffer is to eliminate the possibility of your database being pounded or your e-mail box being flooded.

Definitions of how the buffering should occur are located in the `<bufferModes>` section within the `<healthMonitoring>` section of the `web.config` file. An example of the `<bufferModes>` section is presented in Listing 32-9.

Listing 32-9: Using the <bufferModes> section

```
<configuration>
   <system.web>

      <healthMonitoring enabled="true">
```

```
            <bufferModes>

               <clear />

               <add name="Critical Notification" maxBufferSize="100" maxFlushSize="20"
                urgentFlushThreshold="1" regularFlushInterval="Infinite"
                urgentFlushInterval="00:01:00"
                maxBufferThreads="1" />
               <add name="Notification" maxBufferSize="300" maxFlushSize="20"
                urgentFlushThreshold="1" regularFlushInterval="Infinite"
                urgentFlushInterval="00:01:00"
                maxBufferThreads="1" />
               <add name="Analysis" maxBufferSize="1000" maxFlushSize="100"
                urgentFlushThreshold="100" regularFlushInterval="00:05:00"
                urgentFlushInterval="00:01:00" maxBufferThreads="1" />
               <add name="Logging" maxBufferSize="1000" maxFlushSize="200"
                urgentFlushThreshold="800"
                regularFlushInterval="00:30:00" urgentFlushInterval="00:05:00"
                maxBufferThreads="1" />

            </bufferModes>

            <eventMappings>
               <!-- Code removed for clarity -->
            </eventMappings>

            <providers>
               <!-- Code removed for clarity -->
            </providers>

            <rules>
               <!-- Code removed for clarity -->
            </rules>

            <profiles>
               <!-- Code removed for clarity -->
            </profiles>

         </healthMonitoring>

      </system.web>
   </configuration>
```

In this code, you can see four buffer modes defined. Each mode has a friendly name defined using the `name` attribute. For each mode, a number of different attribute can be applied. To examine these attributes, take a closer look at the `Logging` buffer mode.

```
<add name="Logging"
 maxBufferSize="1000"
 maxFlushSize="200"
 urgentFlushThreshold="800"
 regularFlushInterval="00:30:00"
 urgentFlushInterval="00:05:00"
 maxBufferThreads="1" />
```

This buffer mode is called *Logging* and, based on the values assigned to its attributes, any provider using this buffer mode sends the messages to the database or via e-mail every 30 minutes. This is also referred to as *flushing* the Web events. The time period of 30 minutes is defined using the `regularFlushInterval` attribute. Therefore, every 30 minutes, ASP.NET sends 200 messages to the database (or via e-mail). It will not send more than 200 messages at a time because of what is specified in the `maxFlushSize` attribute. Well, what happens if more than 200 messages are waiting within that 30-minute time period? ASP.NET still sends only 200 messages every 30 minutes and holds additional messages when the number exceeds 200. The maximum number of message held in the queue cannot exceed 1,000 messages. This number is set through the `maxBufferSize` attribute. However, after the total in the queue hits 800 messages, ASP.NET starts flushing the messages every 5 minutes instead of every 30 minutes. The change in frequency of messages is determined by the `urgentFlushThreshold` attribute, and the time interval used to send messages when the `urgentFlushThreshold` is hit is determined by the `urgentFlushInterval` attribute.

After you have defined the buffering modes you want to use, the next step is to apply them. To analyze the process, look back on how the `SqlWebEventProvider` is declared.

```
<providers>

   <clear />

   <add name="EventLogProvider"
    type="System.Web.Management.EventLogWebEventProvider,System.Web,
      Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a" />

   <add name="SqlWebEventProvider"
    connectionStringName="LocalSqlServer"
    maxEventDetailsLength="1073741823"
    buffer="false" bufferMode="Notification"
    type="System.Web.Management.SqlWebEventProvider,System.Web,
      Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a" />

   <add name="WmiWebEventProvider"
    type="System.Web.Management.WmiWebEventProvider,System.Web,
      Version=2.0.0.0,Culture=neutral,PublicKeyToken=b03f5f7f11d50a3a" />

</providers>
```

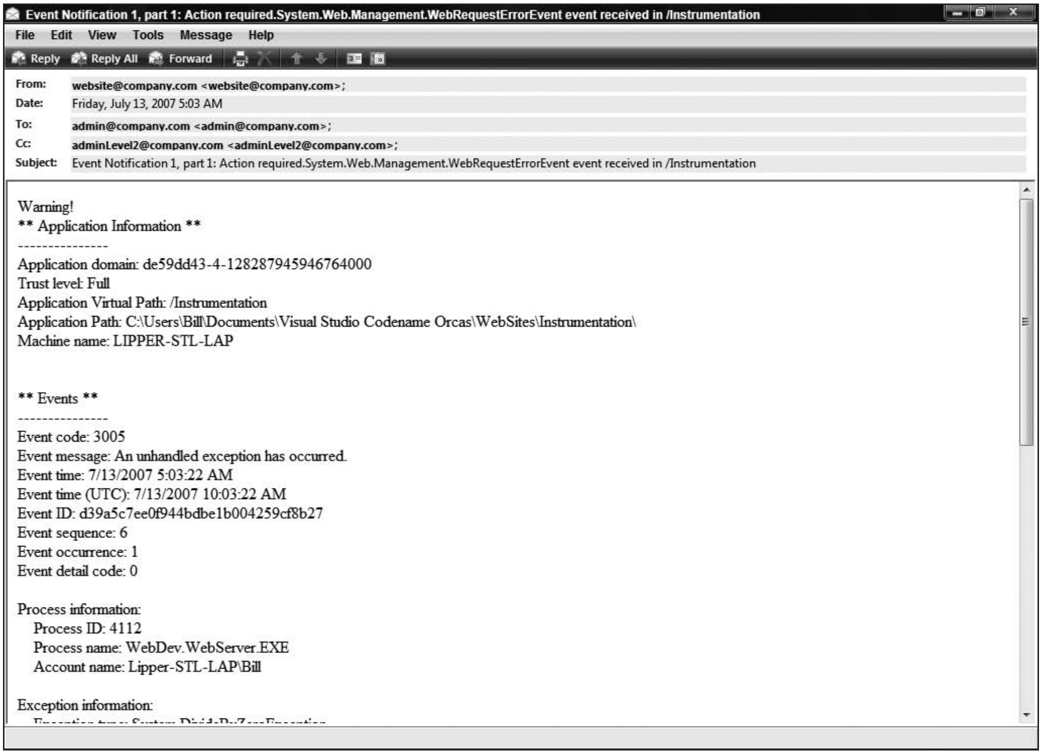
Again, this event is shown in bold. First, the most important attribute is the `buffer` attribute. By default, buffering of messages is turned off. This is done by setting the `buffer` attribute to `false`. The `bufferMode` attribute allows you to assign one of the defined buffer modes created within the `<bufferModes>` section. In this case, the `Notification` buffer mode is referenced. Changing the buffer attribute to `true` enables the events to be sent only to SQL Server according to the time intervals defined by the `Notification` buffer mode.

E-mailing Web Events

When monitoring a server for Web events, you are not *always* going to be networked into a server or actively monitoring it every second. This doesn't mean that you won't want to know immediately if something is going very wrong with your ASP.NET application. For this reason, you will find the `SimpleMailWebEventProvider` and the `TemplatedMailWebEventProvider` objects quite beneficial.

Using the SimpleMailProvider

The `SimpleMailWebEventProvider` sends Web events as a simple text e-mail, as shown in Figure 32-14.


Event Notification 1, part 1: Action required.System.Web.Management.WebRequestErrorEvent event received in /Instrumentation
File
Edit
View
Tools
Message
Help
Reply
Reply All
Forward

From: website@company.com <website@company.com>;
Date: Friday, July 13, 2007 5:03 AM
To: admin@company.com <admin@company.com>;
Cc: adminLevel2@company.com <adminLevel2@company.com>;
Subject: Event Notification 1, part 1: Action required.System.Web.Management.WebRequestErrorEvent event received in /Instrumentation

```
Warning!
** Application Information **
---------------
Application domain: de59dd43-4-128287945946764000
Trust level: Full
Application Virtual Path: /Instrumentation
Application Path: C:\Users\Bill\Documents\Visual Studio Codename Orcas\WebSites\Instrumentation\
Machine name: LIPPER-STL-LAP


** Events **
---------------
Event code: 3005
Event message: An unhandled exception has occurred.
Event time: 7/13/2007 5:03:22 AM
Event time (UTC): 7/13/2007 10:03:22 AM
Event ID: d39a5c7ee0f944bdbe1b004259cf8b27
Event sequence: 6
Event occurrence: 1
Event detail code: 0

Process information:
    Process ID: 4112
    Process name: WebDev.WebServer.EXE
    Account name: Lipper-STL-LAP\Bill

Exception information:
```

Figure 32-14

To set up this provider, you place an additional `<add />` element within the `<providers>` section, as illustrated in Listing 32-10.

Listing 32-10: Adding a SimpleMailWebEventProvider instance

```
<add name="SimpleMailProvider"
 type="System.Web.Management.SimpleMailWebEventProvider, System.Web,
   Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
 from="website@company.com"
 to="admin@company.com"
 cc="adminLevel2@company.com"
 bcc="director@company.com"
 bodyHeader="Warning!"
 bodyFooter="Please investigate ASAP."
 subjectPrefix="Action required."
 buffer="false"
 maxEventLength="4096"
 maxMessagesPerNotification="1" />
```

After the provider is added, you also add a rule that uses this provider in some fashion (as you do with all the other providers). This is done by referencing the `SimpleMailWebEventProvider` name within the provider attribute of the `<add />` element contained in the `<rules>` section.

For this to work, be sure that you set up the SMTP capabilities correctly in your `web.config` file. An example is presented in Listing 32-11.

Listing 32-11: Setting up SMTP in the web.config file

```
<configuration>

  <system.web>
     <!-- Code removed for clarity -->
  </system.web>

  <system.net>
    <mailSettings>
      <smtp deliveryMethod="Network">
        <network
          host="localhost"
          port="25"
          defaultCredentials="true"
        />
      </smtp>
    </mailSettings>
  </system.net>

</configuration>
```

If you do not have a quick way of setting up SMTP and still want to test this, you can also have ASP.NET create e-mail messages and store them to disk by setting up the `<smtp>` section as illustrated in Listing 32-12.

Listing 32-12: Another option for setting up the SMTP section

```
<configuration>

  <system.web>
     <!-- Code removed for clarity -->
  </system.web>

  <system.net>
    <mailSettings>
      <smtp deliveryMethod="SpecifiedPickupDirectory">
        <specifiedPickupDirectory pickupDirectoryLocation ="C:\"/>
      </smtp>
    </mailSettings>
  </system.net>

</configuration>
```

In this scenario, the e-mails will be just placed within the `C:\` root directory.

Using the TemplatedMailWebEventProvider

Another option for e-mailing Web events is to use the `TemplatedMailWebEventProvider` object. This works basically the same as the `SimpleMailWebEventProvider`, but the `TemplatedMailWebEventProvider` allows you to create more handsome e-mails (they might get noticed more). As with the other providers, you use the `<add />` element within the `<providers>` section to add this provider. This process is illustrated in Listing 32-13.

Listing 32-13: Adding a TemplatedMailWebEventProvider

```
<add name="TemplatedMailProvider"
 type="System.Web.Management.TemplatedMailWebEventProvider, System.Web,
   Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
 template="../MailTemplates/ErrorNotification.aspx"
 from="website@company.com"
 to="admin@company.com"
 cc="adminLevel2@company.com"
 bcc="director@company.com"
 subjectPrefix="Action required."
 buffer="false"
 detailedTemplateErrors="true"
 maxMessagesPerNotification="1" />
```

After the provider is added, you also need to add a rule that uses this provider in some fashion (as you do with all the other providers). You add it by referencing the `TemplatedMailWebEventProvider` name within the provider attribute of the `<add />` element contained in the `<rules>` section. Be sure to set up the `<smtp>` section, just as you did with the `SimpleMailWebEventProvider`.

After these items are in place, the next step is to create an `ErrorNotification.aspx` page. This page construction is illustrated in Listing 32-14.

Listing 32-14: Creating the ErrorNotification.aspx page

VB

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Web.Management" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim meni As MailEventNotificationInfo = _
          TemplatedMailWebEventProvider.CurrentNotification
        Label1.Text = "Events Discarded By Buffer: " & _
          meni.EventsDiscardedByBuffer.ToString()
        Label2.Text = "Events Discarded Due To Message Limit: " & _
          meni.EventsDiscardedDueToMessageLimit.ToString()
        Label3.Text = "Events In Buffer: " & meni.EventsInBuffer.ToString()
        Label4.Text = "Events In Notification: " & _
          meni.EventsInNotification.ToString()
        Label5.Text = "Events Remaining: " & meni.EventsRemaining.ToString()
        Label6.Text = "Last Notification UTC: " & _
          meni.LastNotificationUtc.ToString()
        Label7.Text = "Number of Messages In Notification: " & _
```

Continued

```
          meni.MessagesInNotification.ToString()

        DetailsView1.DataSource = meni.Events
        DetailsView1.DataBind()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
   <head></head>
   <body>
      <form id="form1" runat="server">
         <asp:label id="Label1" runat="server"></asp:label><br />
         <asp:label id="Label2" runat="server"></asp:label><br />
         <asp:label id="Label3" runat="server"></asp:label><br />
         <asp:label id="Label4" runat="server"></asp:label><br />
         <asp:label id="Label5" runat="server"></asp:label><br />
         <asp:label id="Label6" runat="server"></asp:label><br />
         <asp:label id="Label7" runat="server"></asp:label><br />

         <br />

         <asp:DetailsView ID="DetailsView1" runat="server" Height="50px"
          Width="500px" BackColor="White" BorderColor="#E7E7FF" BorderStyle="None"
          BorderWidth="1px" CellPadding="3" GridLines="Horizontal">
            <FooterStyle BackColor="#B5C7DE" ForeColor="#4A3C8C" />
            <EditRowStyle BackColor="#738A9C" Font-Bold="True"
             ForeColor="#F7F7F7" />
            <RowStyle BackColor="#E7E7FF" ForeColor="#4A3C8C" />
            <PagerStyle BackColor="#E7E7FF" ForeColor="#4A3C8C"
             HorizontalAlign="Right" />
            <HeaderStyle BackColor="#4A3C8C" Font-Bold="True"
             ForeColor="#F7F7F7" />
            <AlternatingRowStyle BackColor="#F7F7F7" />
         </asp:DetailsView>
      </form>
   </body>
</html>
```

C#

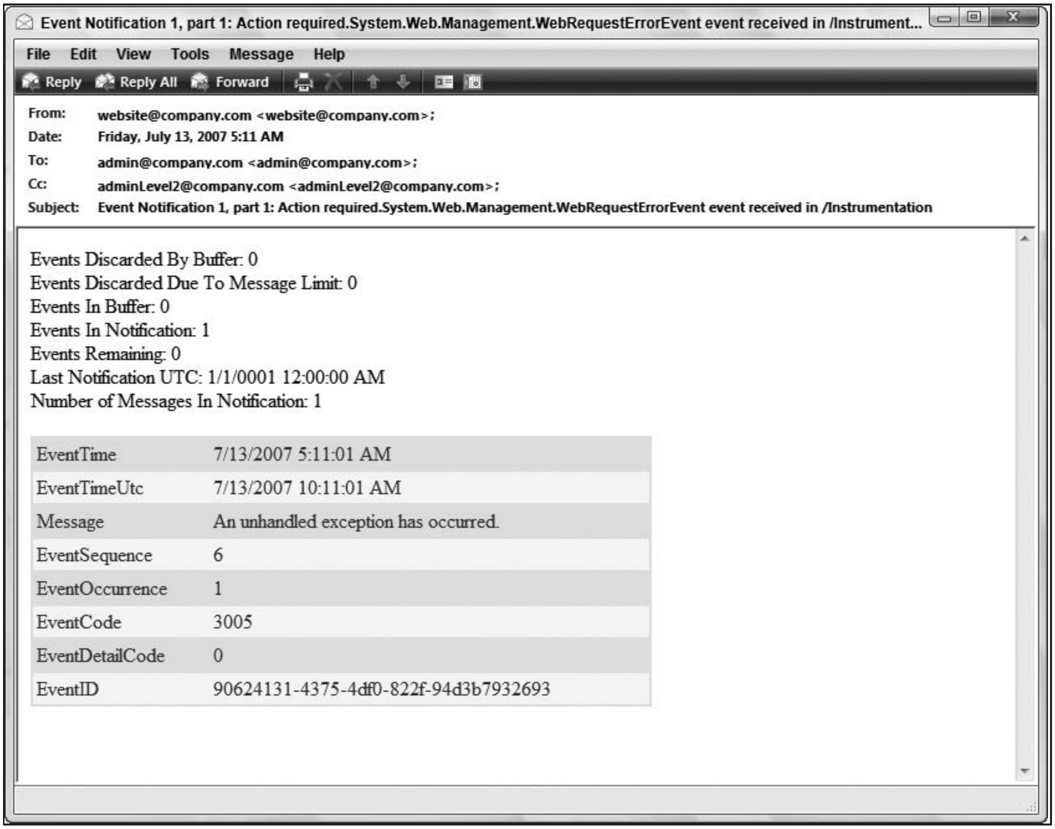
```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Web.Management" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        MailEventNotificationInfo meni =
          TemplatedMailWebEventProvider.CurrentNotification;
        Label1.Text = "Events Discarded By Buffer: " +
          meni.EventsDiscardedByBuffer.ToString();
        Label2.Text = "Events Discarded Due To Message Limit: " +
          meni.EventsDiscardedDueToMessageLimit.ToString();
        Label3.Text = "Events In Buffer: " + meni.EventsInBuffer.ToString();
        Label4.Text = "Events In Notification: " +
          meni.EventsInNotification.ToString();
```

Continued

```
        Label5.Text = "Events Remaining: " + meni.EventsRemaining.ToString();
        Label6.Text = "Last Notification UTC: " +
          meni.LastNotificationUtc.ToString();
        Label7.Text = "Number of Messages In Notification: " +
          meni.MessagesInNotification.ToString();

        DetailsView1.DataSource = meni.Events;
        DetailsView1.DataBind();
    }
</script>
```

To work with the `TemplatedMailWebEventProvider`, you first import the `System.Web.Management` namespace. This is done so you can work with the `MailEventNotificationInfo` and `TemplatedMailWebEventProvider` objects. You first create an instance of the `MailEventNotificationInfo` object and assign it a value of the `TemplatedMailWebEventProvider.CurrentNotfication` property. Now, you have access to an entire series of values from the Web event that was monitored.

This e-mail message is displayed in Figure 32-15.


Event Notification 1, part 1: Action required.System.Web.Management.WebRequestErrorEvent event received in /Instrument...
File
Edit
View
Tools
Message
Help
Reply
Reply All
Forward

From: website@company.com <website@company.com>;
Date: Friday, July 13, 2007 5:11 AM
To: admin@company.com <admin@company.com>;
Cc: adminLevel2@company.com <adminLevel2@company.com>;
Subject: Event Notification 1, part 1: Action required.System.Web.Management.WebRequestErrorEvent event received in /Instrumentation

Events Discarded By Buffer: 0
Events Discarded Due To Message Limit: 0
Events In Buffer: 0
Events In Notification: 1
Events Remaining: 0
Last Notification UTC: 1/1/0001 12:00:00 AM
Number of Messages In Notification: 1

EventTime	7/13/2007 5:11:01 AM
EventTimeUtc	7/13/2007 10:11:01 AM
Message	An unhandled exception has occurred.
EventSequence	6
EventOccurrence	1
EventCode	3005
EventDetailCode	0
EventID	90624131-4375-4df0-822f-94d3b7932693

Figure 32-15

As you can see in this figure, the e-mail message is more readable in this format.

Summary

Whereas ASP.NET 1.*x* was really focused on the developer, ASP.NET 2.0 and 3.5 have made tremendous inroads into making life easier for the administrator of the deployed ASP.NET application. In addition to a number of GUI-based management and administration tools (covered in the next chapter), you can now record and send notifications about the health (good or bad) of your ASP.NET applications using the ASP.NET health monitoring capabilities.

33

Administration and Management

You have almost reached the end of this book; you have been introduced to ASP.NET 3.5 with its wonderful features designed to help you become a better and more efficient programmer. However, with all advancement comes complexity, as is the case in the areas of ASP.NET configuration and management. The good news is that the ASP.NET development team realized this and provided tools and APIs that enable developers to configure and manage ASP.NET–based applications with reliability and comfort.

This chapter covers these tools in great detail in an effort to educate you about some of the options available to you. This chapter explores two powerful configuration tools: the ASP.NET Web Site Administration Tool, a Web-based application, and the IIS Manager, which is used to configure your ASP.NET applications.

The ASP.NET Web Site Administration Tool

When ASP.NET was first released, it introduced the concept of an XML-based configuration file for its Web applications. This `web.config` file is located in the same directory as the application itself. It is used to store a number of configuration settings, some of which can override configuration settings defined in `machine.config` file or in the root server's `web.config` file. Versions of ASP.NET before ASP.NET 2.0, however, did not provide an administration tool to make it easy to configure the settings. Because of this, a large number of developers around the world ended up creating their own configuration tools to avoid having to work with the XML file manually.

The ASP.NET Web Site Administration Tool enables you to manage Web site configuration through a simple, easy-to-use Web interface. It eliminates the need for manually editing the `web.config` file. If no `web.config` file exists when you use the administration tool for the first time, it creates one. By default, the ASP.NET Web Site Administration Tool also creates the standard `ASPNETDB.MDF` SQL Server Express Edition file in the `App_Data` folder of your Web site to store application data.

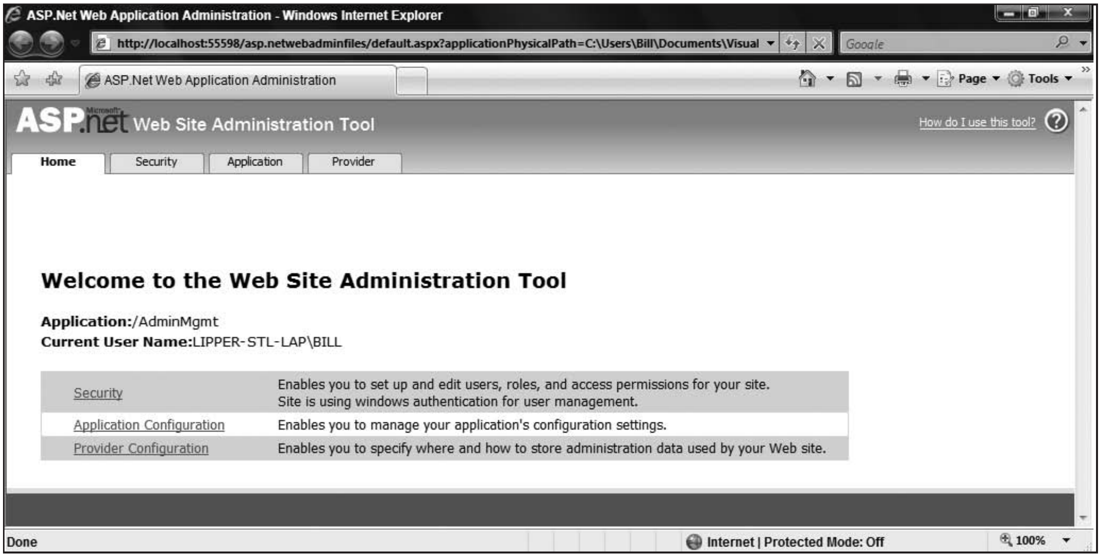
The changes made to most settings in the ASP.NET Web Site Administration Tool take effect immediately. You find tm reflected in the `web.config` file.

The default settings are automatically inherited from any configuration files that exist in the root folder of a Web server. The ASP.NET Web Site Administration Tool enables you to create or update your own settings for your Web application. You can also override the settings inherited from uplevel configuration files, if an override for those settings is allowed. If overriding is not permitted, the setting appears dimmed in the administration tool.

The ASP.NET Web Site Administration Tool is automatically installed during installation of the .NET Framework version 3.5. To use the administration tool to administer your own Web site, you must be logged in as a registered user of your site and you must have read and write permissions to `web.config`.

You cannot access the ASP.NET Web Site Administration Tool remotely or even locally through IIS. Instead, you access it with Visual Studio 2008, which, in turn, uses its integrated web server (formally named Cassini) to access the administration tool.

In order to access this tool through Visual Studio 2008, open the website and click the ASP.NET Configuration button found in the menu located at the top of the Solution Explorer pane. Another way to launch this tool is to select ASP.NET Configuration from the Website option in the main Visual Studio menu. Figure 33-1 shows the ASP.NET Web Site Administration Tool's welcome page.


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/default.aspx?applicationPhysicalPath=C:\Users\Bill\Documents\Visual
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Done
Internet | Protected Mode: Off
100%

Welcome to the Web Site Administration Tool

Application:/AdminMgmt
Current User Name:LIPPER-STL-LAP\BILL

Security	Enables you to set up and edit users, roles, and access permissions for your site. Site is using windows authentication for user management.
Application Configuration	Enables you to manage your application's configuration settings.
Provider Configuration	Enables you to specify where and how to store administration data used by your Web site.

Figure 33-1

The ASP.NET Web Site Administration Tool features a tabbed interface that groups related configuration settings. The tabs and the configuration settings that they manage are described in the following sections.

The Home Tab

The Home tab (shown previously in Figure 33-1) is a summary that supplies some basic information about the application you are monitoring or modifying. It provides the name of the application and the current user context in which you are accessing the application. In addition, you see links to the other administration tool tabs that provide you with summaries of their settings. To make any changes to your Web application, you simply click the appropriate tab or link.

Remember that most changes to configuration settings made using this administration tool take effect immediately, causing the Web application to be restarted and currently active sessions to be lost if you are using an InProc session. The best practice for administrating ASP.NET is to make configuration changes to a development version of your application and later publish these changes to your production application. That's why this tool can't be used outside of Visual Studio.

Some settings (those in which the administration tool interface has a dedicated Save button) do not save automatically. You can lose the information typed in these windows if you do not click the Save button to propagate the changes you made to the `web.config` file. The ASP.NET Web Site Administration Tool also times out after a period of inactivity. Any settings that do not take effect immediately and are not saved will be lost if this occurs.

As extensive as the ASP.NET Web Site Administration Tool is, it manages only some of the configuration settings that are available for your Web application. All other settings require modification of configuration files manually, by using the Microsoft Management Console (MMC) snap-in for ASP.NET if you are using Windows XP, using the Internet Information Services (IIS) Manager if you are using Windows Vista, or by using the `Configuration` API.

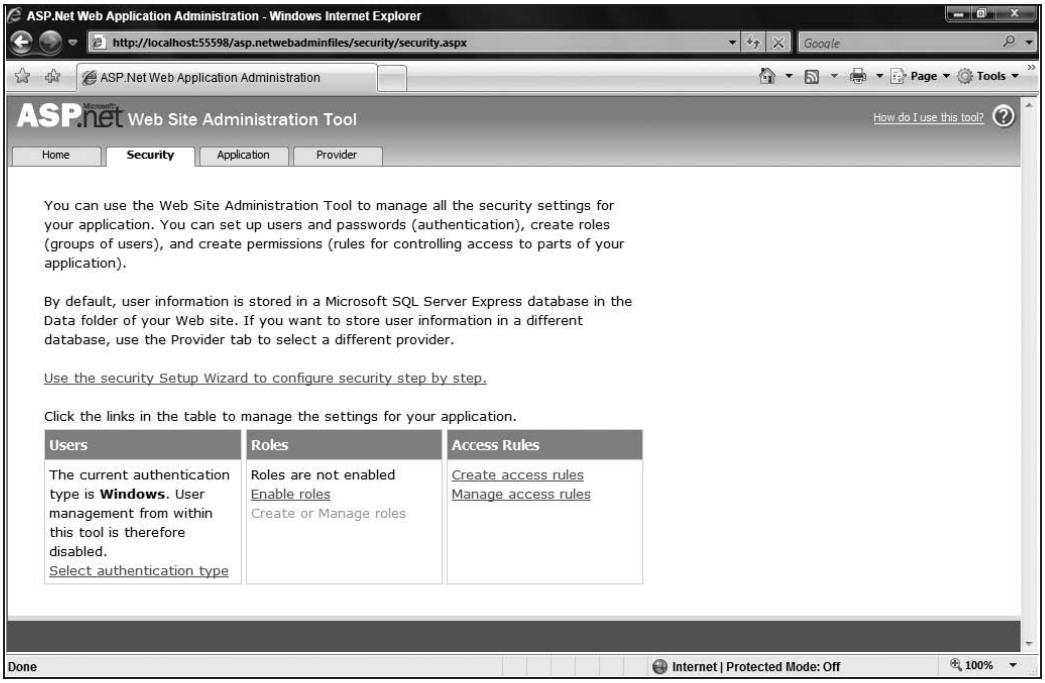
The Security Tab

Use the Security tab to manage access permissions to secure sections of your Web application, user accounts, and roles. From this tab, you can select whether your Web application is accessed on an intranet or from the Internet. If you specify the intranet, Windows-based authentication is used; otherwise, forms-based authentication is configured. The latter mechanism relies on you to manage users in a custom data store, such as SQL Server database tables. The Windows-based authentication employs the user's Windows logon for identification.

User information is stored in a SQL Server Express database by default (`ASPNETDB.MDF`). The database is automatically created in the `App_Data` folder of the Web application. It is recommended that you store such sensitive information on a different and more secure database, perhaps located on a separate server. Changing the data store might mean that you also need to change the underlying data provider. To accomplish this, you simply use the Provider tab to select a different data provider. The Provider tab is covered later in this chapter.

You can configure security settings on this tab in two ways: select the Setup Wizard, or simply use the links provided for the Users, Roles, and Access Management sections. Figure 33-2 shows the Security tab.

You can use the wizard to configure initial settings. Later, you learn other ways to create and modify security settings.


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/security/security.aspx
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Done
Internet | Protected Mode: Off
100%

You can use the Web Site Administration Tool to manage all the security settings for your application. You can set up users and passwords (authentication), create roles (groups of users), and create permissions (rules for controlling access to parts of your application).

By default, user information is stored in a Microsoft SQL Server Express database in the Data folder of your Web site. If you want to store user information in a different database, use the Provider tab to select a different provider.

Use the security Setup Wizard to configure security step by step.

Click the links in the table to manage the settings for your application.

Users	Roles	Access Rules
The current authentication type is **Windows**. User management from within this tool is therefore disabled. Select authentication type	Roles are not enabled Enable roles Create or Manage roles	Create access rules Manage access rules

Figure 33-2

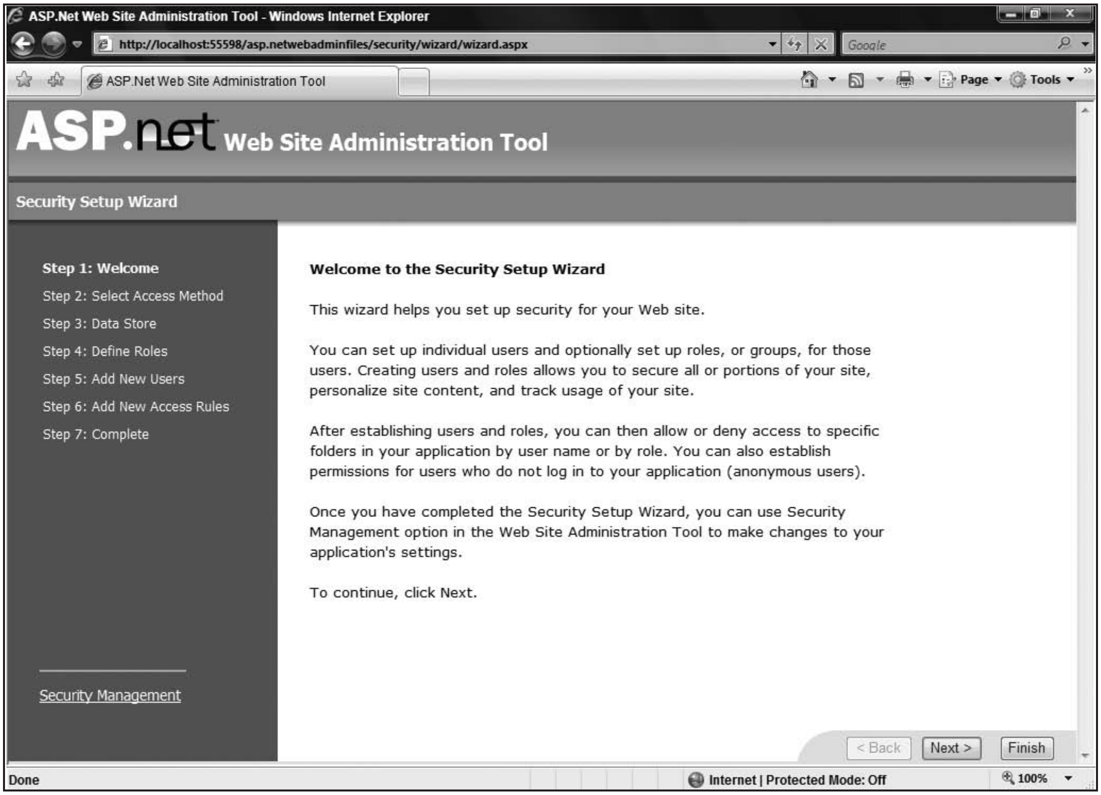
The Security Setup Wizard

The Security Setup Wizard provides a seven-step process ranging from selecting the way the user will be authenticated to selecting a data source for storing user information. This is followed by definitions of roles, users, and access rules.

> **Be sure to create all folders that need special permissions *before* you engage the wizard.**

Follow these steps to use the Security Setup Wizard:

1. The wizard welcome screen (shown in Figure 33-3) is informational only. It educates you on the basics of security management in ASP.NET. When you finish reading the screen, click Next.
2. Select your access method (authentication mechanism). You have two options:
   - **From the Internet:** Indicates you want forms-based authentication. You must use your own database of user information. This option works well in scenarios where non-employees need to access the Web application.
   - **From a Local Area Network:** Indicates users of this application are already authenticated on the domain. You do not have to use your own user information database. Instead, you can use the Windows web server domain user information.
   Figure 33-4 shows the screen for Step 2 of the process.


ASP.Net Web Site Administration Tool - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/security/wizard/wizard.aspx
Google
ASP.Net Web Site Administration Tool
Page
Tools
ASP.net Web Site Administration Tool
Security Setup Wizard
Step 1: Welcome
Step 2: Select Access Method
Step 3: Data Store
Step 4: Define Roles
Step 5: Add New Users
Step 6: Add New Access Rules
Step 7: Complete
Security Management
< Back
Next >
Finish
Done
Internet | Protected Mode: Off
100%

Welcome to the Security Setup Wizard

This wizard helps you set up security for your Web site.

You can set up individual users and optionally set up roles, or groups, for those users. Creating users and roles allows you to secure all or portions of your site, personalize site content, and track usage of your site.

After establishing users and roles, you can then allow or deny access to specific folders in your application by user name or by role. You can also establish permissions for users who do not log in to your application (anonymous users).

Once you have completed the Security Setup Wizard, you can use Security Management option in the Web Site Administration Tool to make changes to your application's settings.

To continue, click Next.

Figure 33-3

Select **From the Internet**, and click the Next button.

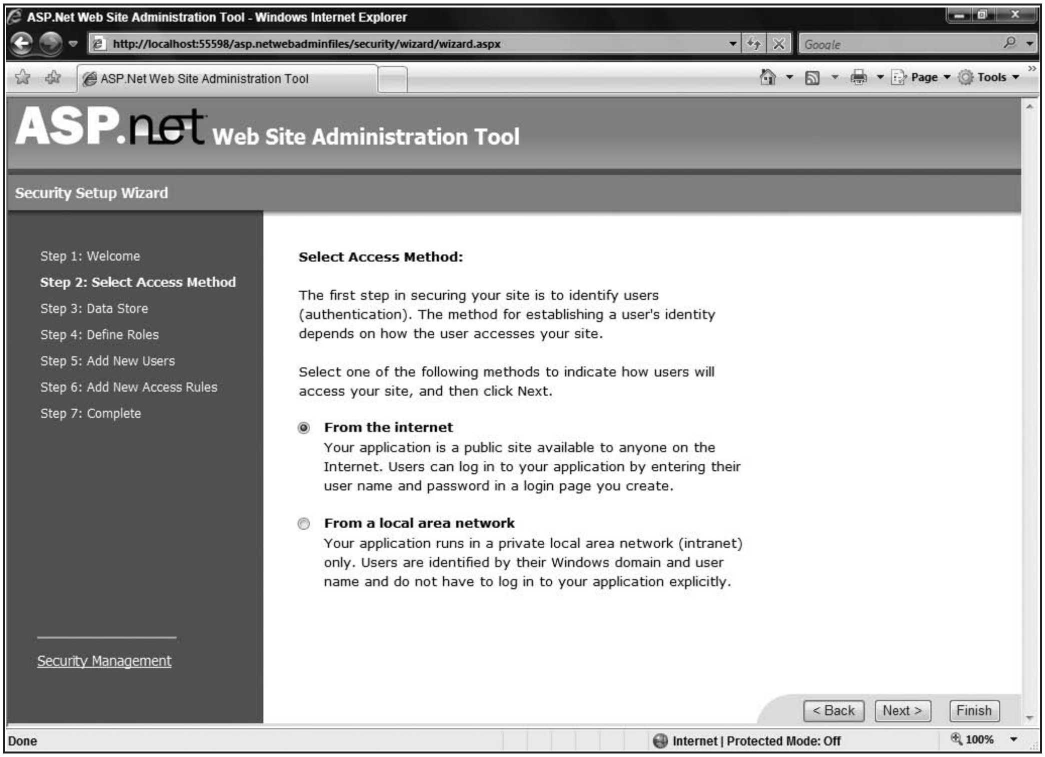
3. Select Access Method. As mentioned earlier, the ASP.NET Web Site Administration Tool uses SQL Server Express Edition by default. You can configure additional providers on the Providers tab. In the Step 3 screen shown in Figure 33-5, only an advanced provider is displayed because no other providers have been configured yet. Click Next.

4. Define Roles. If you are happy with all users having the same access permission, you can simply skip this step by deselecting the Enable Roles for This Web Site check box. If this box is not checked, clicking the Next button takes you directly to the User Management screens. Check this box to see how to define roles using this wizard.

   The screen from Step 4 is shown in Figure 33-6. When you are ready, click Next.

   The next screen (see Figure 33-7) in the wizard enables you to create and delete roles. The roles simply define categories of users. Later, you can provide users and access rules based on these roles. Go ahead and create roles for Administrator, Human Resources, Sales, and Viewer. Click Next.

5. Add New Users. Earlier, you selected the From the Internet option, so the wizard assumes that you want to use forms authentication and provides you with the option of creating and managing users. The From a Local Area Network option, remember, uses Windows-based authentication.


ASP.Net Web Site Administration Tool - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/security/wizard/wizard.aspx
Google
ASP.Net Web Site Administration Tool
Page
Tools
ASP.net Web Site Administration Tool
Security Setup Wizard
Step 1: Welcome
Step 2: Select Access Method
Step 3: Data Store
Step 4: Define Roles
Step 5: Add New Users
Step 6: Add New Access Rules
Step 7: Complete
Security Management
< Back
Next >
Finish
Done
Internet | Protected Mode: Off
100%

Select Access Method:

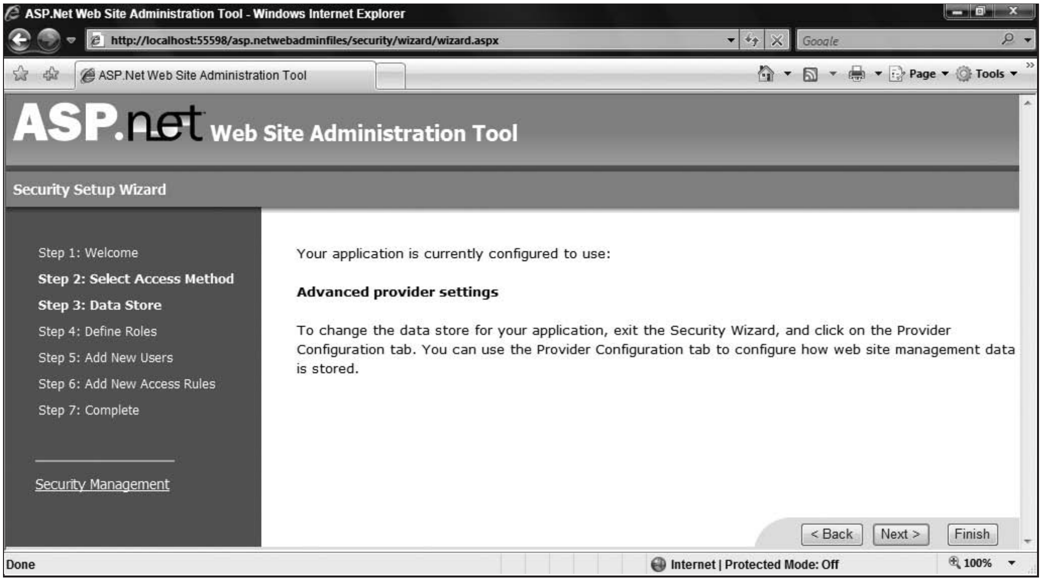
The first step in securing your site is to identify users (authentication). The method for establishing a user's identity depends on how the user accesses your site.

Select one of the following methods to indicate how users will access your site, and then click Next.

● **From the internet**
Your application is a public site available to anyone on the Internet. Users can log in to your application by entering their user name and password in a login page you create.

○ **From a local area network**
Your application runs in a private local area network (intranet) only. Users are identified by their Windows domain and user name and do not have to log in to your application explicitly.

Figure 33-4

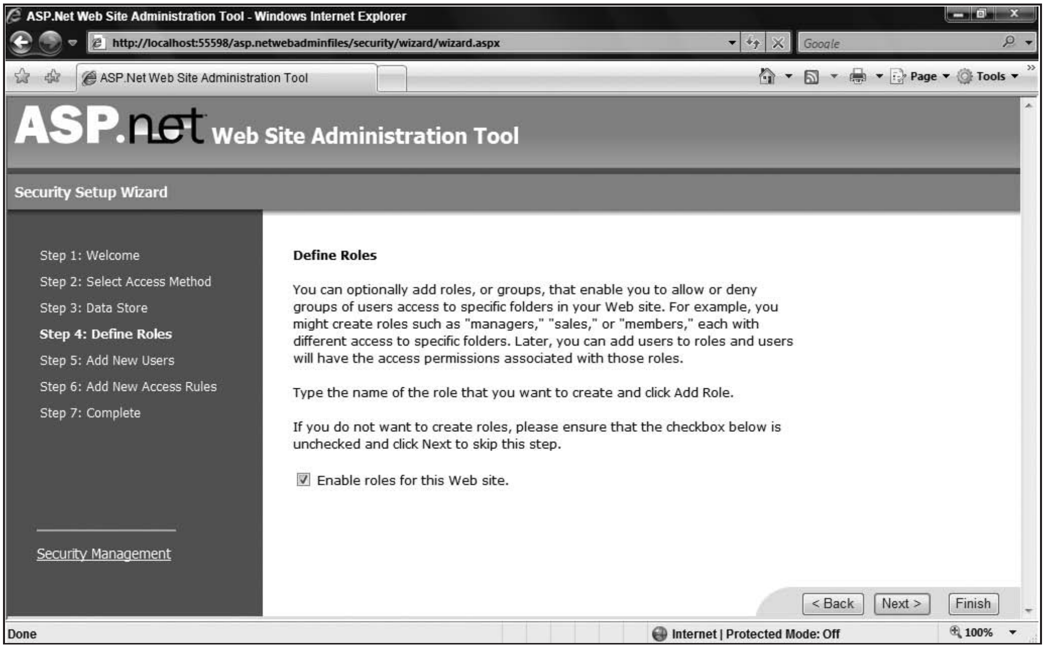

ASP.Net Web Site Administration Tool - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/security/wizard/wizard.aspx
Google
ASP.Net Web Site Administration Tool
Page
Tools
ASP.net Web Site Administration Tool
Security Setup Wizard
Step 1: Welcome
Step 2: Select Access Method
Step 3: Data Store
Step 4: Define Roles
Step 5: Add New Users
Step 6: Add New Access Rules
Step 7: Complete
Security Management
< Back
Next >
Finish
Done
Internet | Protected Mode: Off
100%

Your application is currently configured to use:

Advanced provider settings

To change the data store for your application, exit the Security Wizard, and click on the Provider Configuration tab. You can use the Provider Configuration tab to configure how web site management data is stored.

Figure 33-5


ASP.Net Web Site Administration Tool - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/security/wizard/wizard.aspx
Google
ASP.Net Web Site Administration Tool
Page
Tools
ASP.net Web Site Administration Tool
Security Setup Wizard
Step 1: Welcome
Step 2: Select Access Method
Step 3: Data Store
Step 4: Define Roles
Step 5: Add New Users
Step 6: Add New Access Rules
Step 7: Complete
Security Management
< Back
Next >
Finish
Done
Internet | Protected Mode: Off
100%

Define Roles

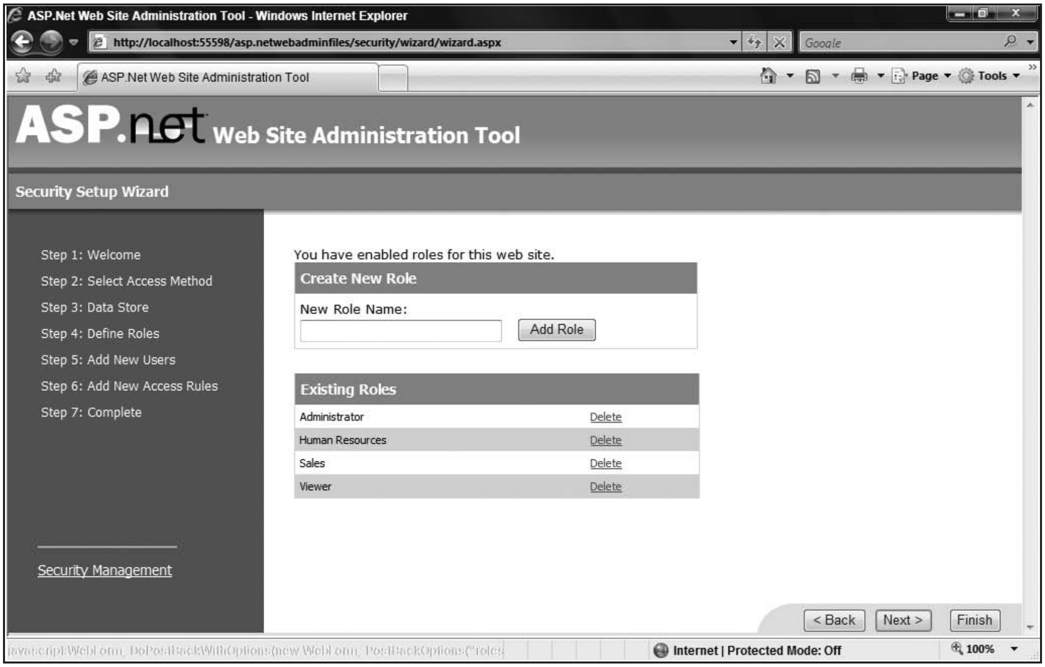
You can optionally add roles, or groups, that enable you to allow or deny groups of users access to specific folders in your Web site. For example, you might create roles such as "managers," "sales," or "members," each with different access to specific folders. Later, you can add users to roles and users will have the access permissions associated with those roles.

Type the name of the role that you want to create and click Add Role.

If you do not want to create roles, please ensure that the checkbox below is unchecked and click Next to skip this step.

☑ Enable roles for this Web site.

Figure 33-6

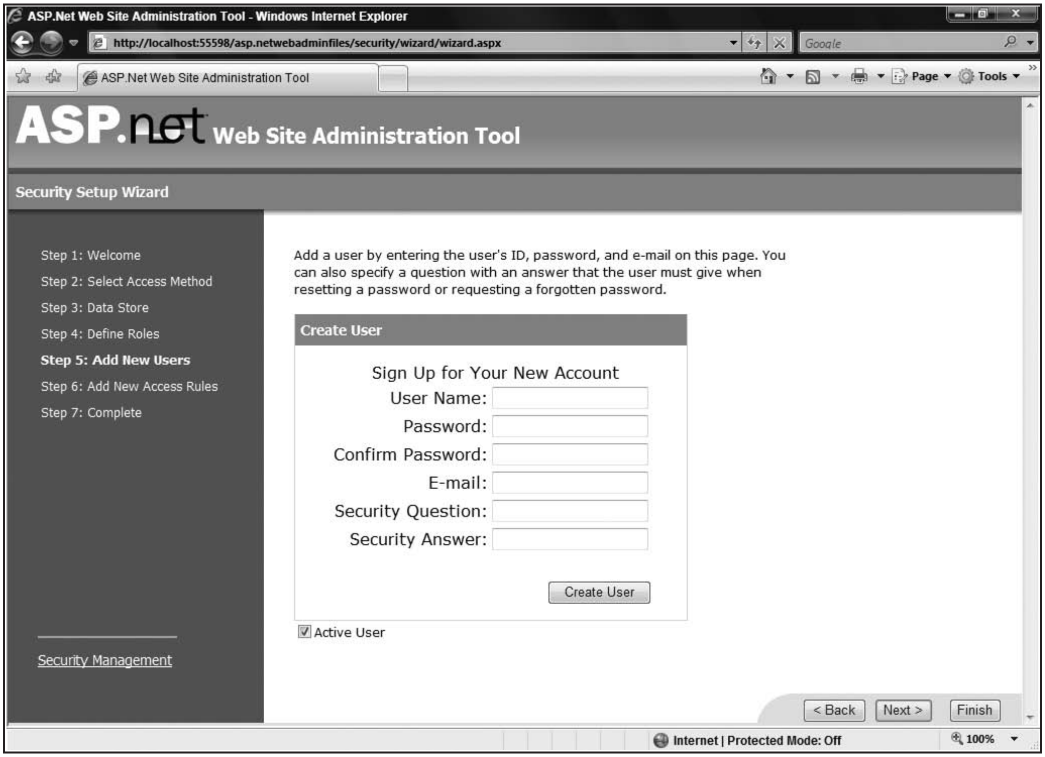

ASP.Net Web Site Administration Tool - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/security/wizard/wizard.aspx
Google
ASP.Net Web Site Administration Tool
Page
Tools
ASP.net Web Site Administration Tool
Security Setup Wizard
Step 1: Welcome
Step 2: Select Access Method
Step 3: Data Store
Step 4: Define Roles
Step 5: Add New Users
Step 6: Add New Access Rules
Step 7: Complete
Security Management
Create New Role
New Role Name:
Add Role
< Back
Next >
Finish
javascript:WebForm_DoPostBackWithOptions(new WebForm_PostBackOptions("roles
Internet | Protected Mode: Off
100%

You have enabled roles for this web site.

Existing Roles	
Administrator	Delete
Human Resources	Delete
Sales	Delete
Viewer	Delete

Figure 33-7

The Add New Users screen (see Figure 33-8) enables you to enter the username, password, e-mail address, and a security question and answer.


ASP.Net Web Site Administration Tool - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/security/wizard/wizard.aspx
Google
ASP.Net Web Site Administration Tool
Page
Tools
ASP.net Web Site Administration Tool
Security Setup Wizard
Step 1: Welcome
Step 2: Select Access Method
Step 3: Data Store
Step 4: Define Roles
Step 5: Add New Users
Step 6: Add New Access Rules
Step 7: Complete
Security Management
Create User
Sign Up for Your New Account
User Name:
Password:
Confirm Password:
E-mail:
Security Question:
Security Answer:
Create User
< Back
Next >
Finish
Internet | Protected Mode: Off
100%

Add a user by entering the user's ID, password, and e-mail on this page. You can also specify a question with an answer that the user must give when resetting a password or requesting a forgotten password.

☑ Active User

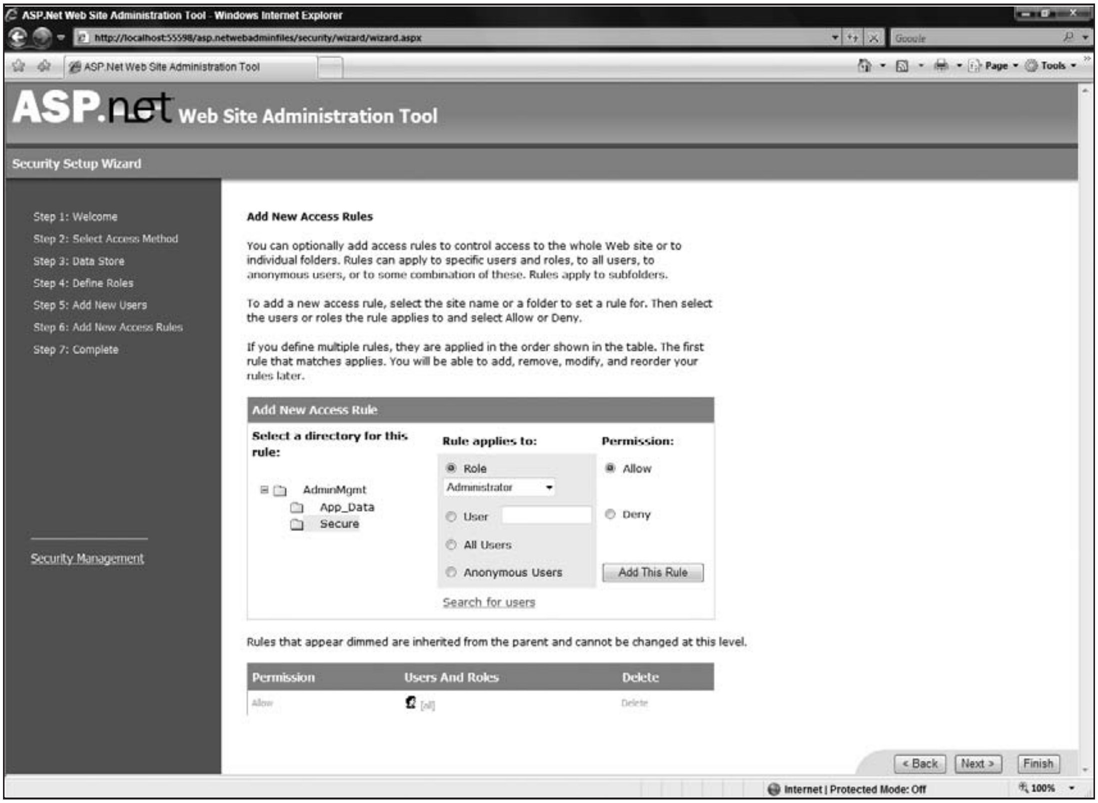
Figure 33-8

You can create as many users as you like; but to delete or update information for users, you must leave the wizard and manage the users separately. As mentioned earlier, the wizard is simply for creating the initial configuration for future management.
Click Next.

6. Add New Access Rules (see Figure 33-9). First, select the folder in the Web application that needs special security settings. Then choose the role or user(s) to whom the rule will apply. Select the permission (Allow or Deny) and click the Add This Rule button. For example, if you had a folder named `Secure` you could select it and the Administrator role, and then click the Allow radio button to permit all users in the Administrator role to access to the `Secure` folder.

> *All folders that need special permissions must be created ahead of time. The information shown in the wizard is cached and is not updated if you decide to create a new folder inside your Web application while you are already on this screen so remember to create your special security folders before starting the wizard.*

The wizard gives you the capability to apply access rules to either roles or specific users. The Search for Users option is handy if you have defined many users for your Web site and want to search for a specific user.


ASP.Net Web Site Administration Tool - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/security/wizard/wizard.aspx
ASP.Net Web Site Administration Tool
ASP.net Web Site Administration Tool
Security Setup Wizard
Step 1: Welcome
Step 2: Select Access Method
Step 3: Data Store
Step 4: Define Roles
Step 5: Add New Users
Step 6: Add New Access Rules
Step 7: Complete
Security Management
Add New Access Rule
AdminMgmt
App_Data
Secure
Add This Rule
Search for users
Permission
Users And Roles
Delete
< Back
Next >
Finish
Internet | Protected Mode: Off
100%

Add New Access Rules

You can optionally add access rules to control access to the whole Web site or to individual folders. Rules can apply to specific users and roles, to all users, to anonymous users, or to some combination of these. Rules apply to subfolders.

To add a new access rule, select the site name or a folder to set a rule for. Then select the users or roles the rule applies to and select Allow or Deny.

If you define multiple rules, they are applied in the order shown in the table. The first rule that matches applies. You will be able to add, remove, modify, and reorder your rules later.

Select a directory for this rule: Secure

Rule applies to: ● Role: Administrator ○ User ○ All Users ○ Anonymous Users

Permission: ● Allow ○ Deny

Rules that appear dimmed are inherited from the parent and cannot be changed at this level.

Permission	Users And Roles	Delete
Allow	[all]	Delete

Figure 33-9

All access rules are shown at the bottom on the screen, and you can delete a specific rule and start again. Rules are shown dimmed if they are inherited from the parent configuration and cannot be changed here.

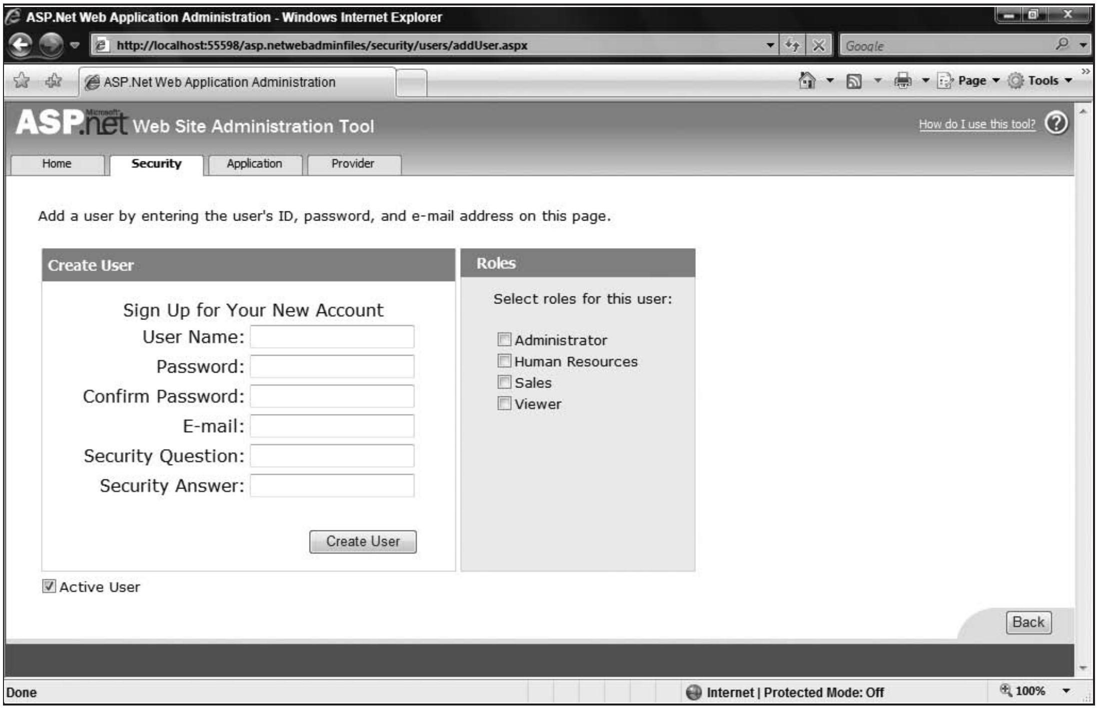
When you are ready, click Next.

7. The last screen in the Security Setup Wizard is an information page. Click the Finish button to exit the wizard.

Creating New Users

The ASP.NET Web Site Administration Tool's Security tab provides ways to manage users without using the wizard and is very helpful for ongoing maintenance of users, roles, and access permissions.

To create a new user, simply click the Create User link on the main page of the Security tab (as you saw earlier in Figure 33-2). The Create User screen, shown in Figure 33-10, is displayed, enabling you to provide username, password, confirmation of password, e-mail, and the security question and answer. You can assign a new user to any number of roles in the Roles list; these are roles currently defined for your Web application. Use this tool to create users named Admin, HRUser and SalesUser and assign them the corresponding roles.


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/security/users/addUser.aspx
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Create User
Roles
Create User
Back
Done
Internet | Protected Mode: Off
100%

Add a user by entering the user's ID, password, and e-mail address on this page.

Sign Up for Your New Account

User Name:

Password:

Confirm Password:

E-mail:

Security Question:

Security Answer:

☑ Active User

Select roles for this user:

☐ Administrator
☐ Human Resources
☐ Sales
☐ Viewer

Figure 33-10

Managing Users

You can manage existing users by clicking the Manage Users link on the Security tab. A new screen displays a list of all existing users (see Figure 33-11). A search option is available, which makes it easier to find a specific user if the list is long.

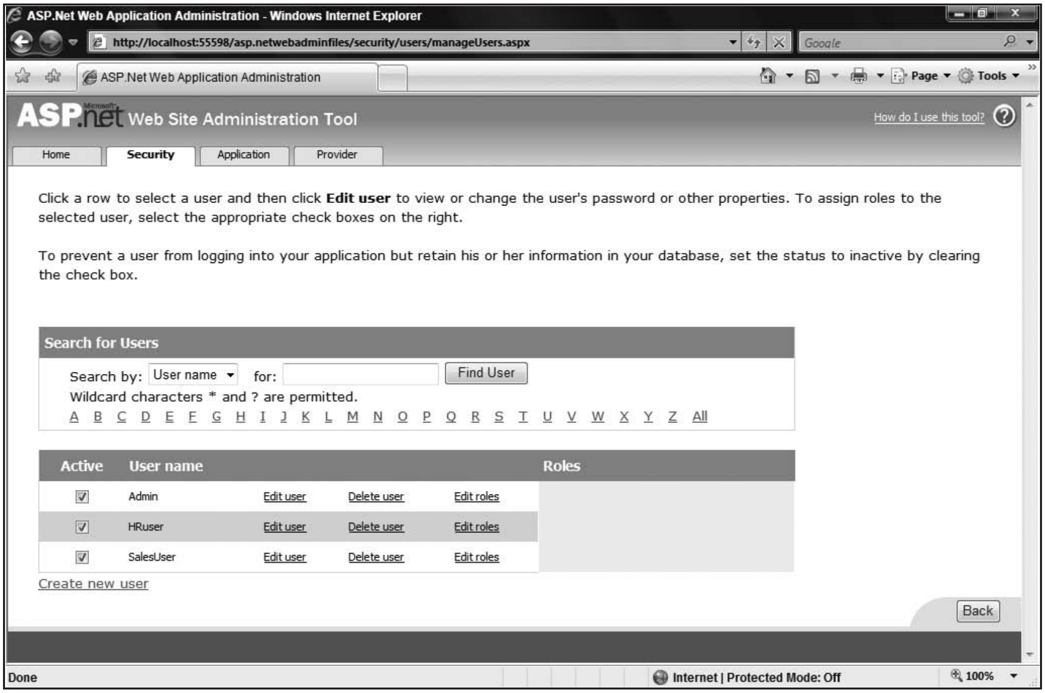
Find the user you want to manage, then you can update his information, delete the user, reassign roles, or set the user to active or inactive.

Managing Roles

Two links are provided in the Security tab for managing roles: Disable Roles and Create or Manage Roles. Clicking Disable Roles does just that — disables role management in the Web application; it also dims the other link.

Click the Create or Manage Roles link to start managing roles and user assignments to specific roles. A screen displays all roles you have defined so far. You have options to add new roles, delete existing roles, or manage specific roles.

Click the Manage link next to a specific role, and a screen shows all the users currently assigned to that role (see Figure 33-12). You can find other users by searching for their names, and you can then assign them to or remove them from a selected role.


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/security/users/manageUsers.aspx
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Find User
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z All
Create new user
Back
Done
Internet | Protected Mode: Off
100%

Click a row to select a user and then click **Edit user** to view or change the user's password or other properties. To assign roles to the selected user, select the appropriate check boxes on the right.

To prevent a user from logging into your application but retain his or her information in your database, set the status to inactive by clearing the check box.

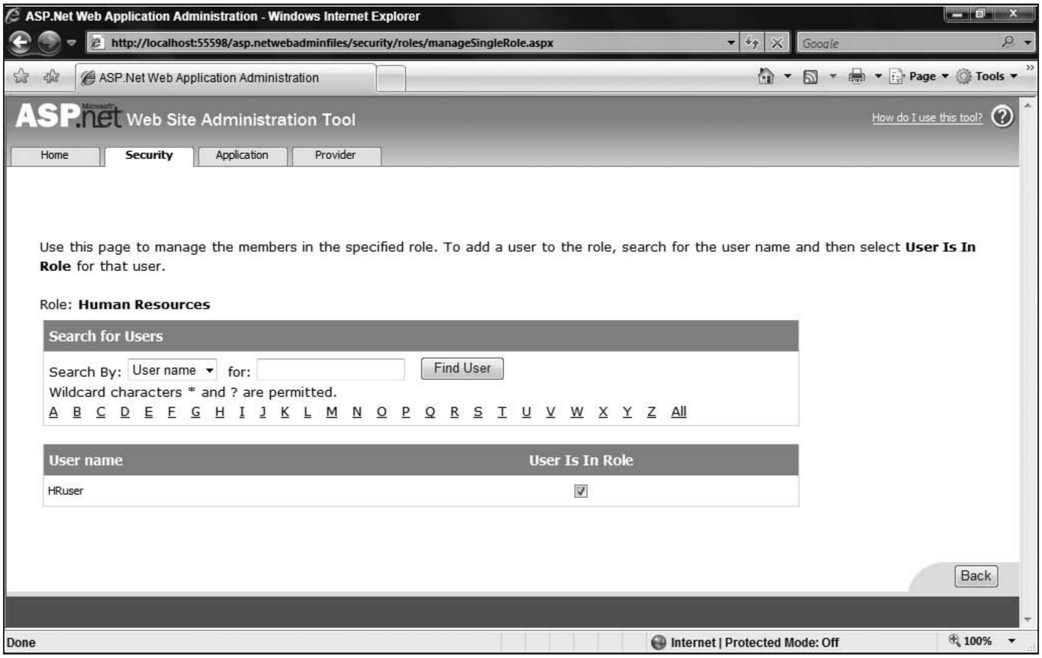
Search for Users

Search by: User name for:

Wildcard characters * and ? are permitted.

Active	User name				Roles
☑	Admin	Edit user	Delete user	Edit roles	
☑	HRuser	Edit user	Delete user	Edit roles	
☑	SalesUser	Edit user	Delete user	Edit roles	

Figure 33-11


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/security/roles/manageSingleRole.aspx
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Find User
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z All
Back
Done
Internet | Protected Mode: Off
100%

Use this page to manage the members in the specified role. To add a user to the role, search for the user name and then select **User Is In Role** for that user.

Role: **Human Resources**

Search for Users

Search By: User name for:

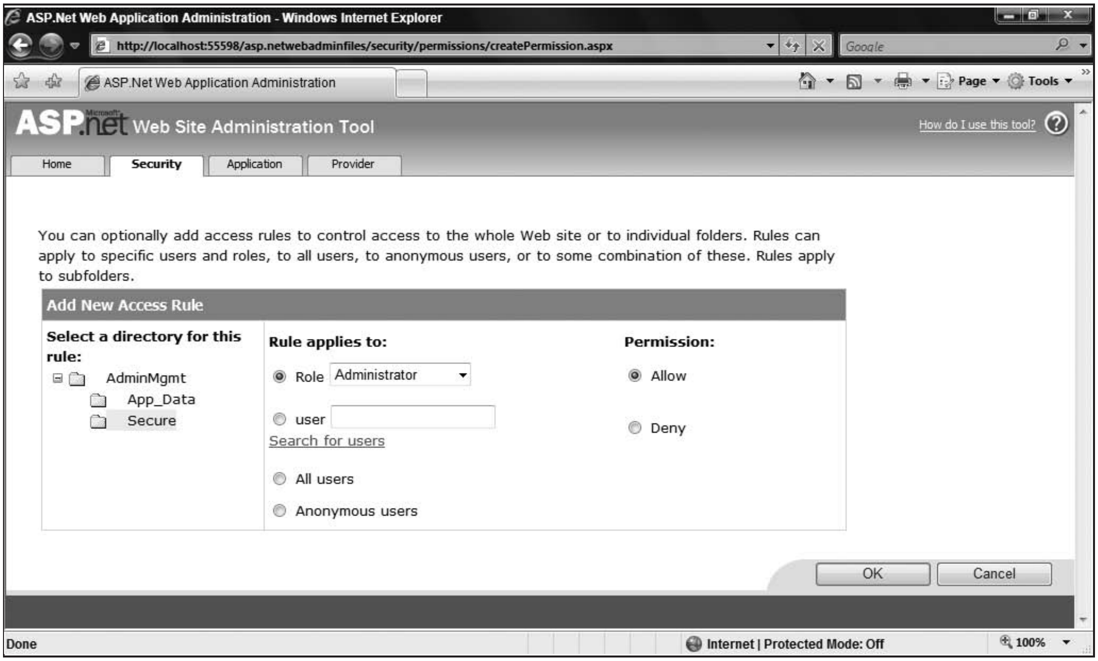
Wildcard characters * and ? are permitted.

User name	User Is In Role
HRuser	☑

Figure 33-12

Managing Access Rules

The Security tab provides options for creating and managing access rules. Access rules are applied either to an entire Web application or to specific folders inside it. Clicking the Create Access Rules link takes you to a screen where you can view a list of the folders inside your Web application. You can select a specific folder, select a role or a user, and then choose whether you want to enable access to the selected folder. Figure 33-13 shows the Add New Access Rule screen.


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/security/permissions/createPermission.aspx
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
OK
Cancel
Done
Internet | Protected Mode: Off
100%

You can optionally add access rules to control access to the whole Web site or to individual folders. Rules can apply to specific users and roles, to all users, to anonymous users, or to some combination of these. Rules apply to subfolders.

Add New Access Rule

Select a directory for this rule:

- AdminMgmt
  - App_Data
  - Secure

Rule applies to:

● Role Administrator

○ user

Search for users

○ All users

○ Anonymous users

Permission:

● Allow

○ Deny

Figure 33-13

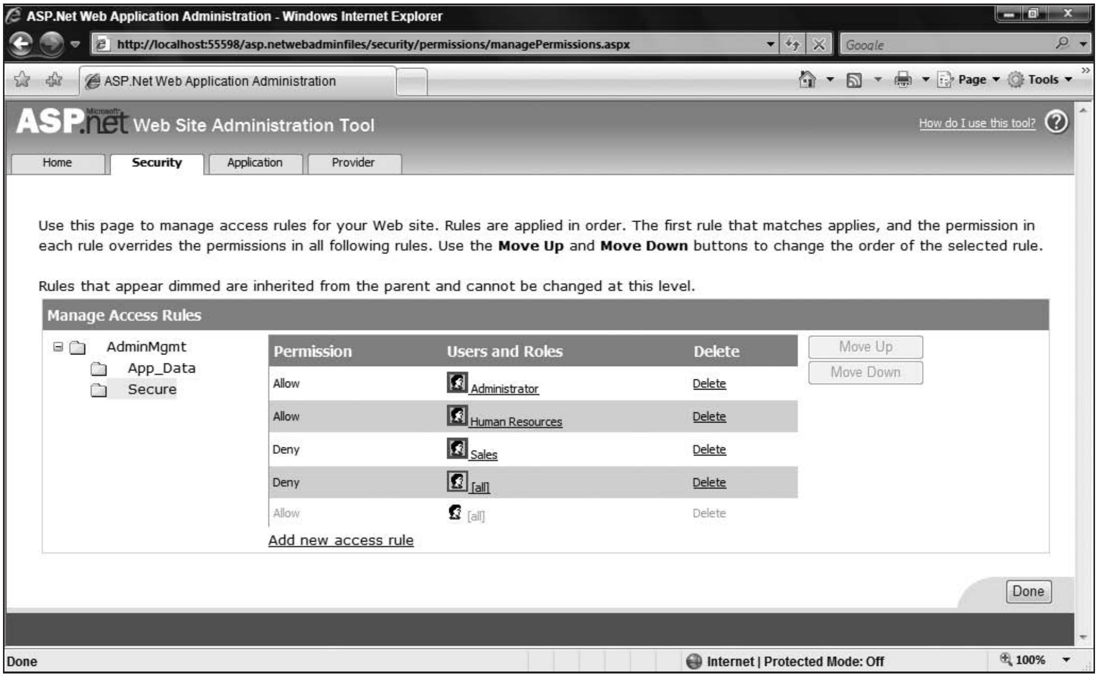
Clicking Manage Access Rules on the Security tab takes you to a screen that shows all existing access rules. You can remove any of these rules and add new ones. You can also readjust the list of access rules if you want to apply them in a specific order. The Manage Access Rules screen is shown in Figure 33-14.

The Application Tab

The Application tab provides a number of application-specific configurations, including the configuration of appSettings, SMTP mail server settings, debugging and trace settings, and starting/stopping the entire Web application.

Managing Application Settings

The left side of the screen shows links for creating and managing application settings. The settings are stored in the `<appSettings>` section of the `web.config`. Most ASP.NET programmers


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/security/permissions/managePermissions.aspx
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Manage Access Rules
AdminMgmt
App_Data
Secure
Move Up
Move Down
Add new access rule
Done
Done
Internet | Protected Mode: Off
100%

Use this page to manage access rules for your Web site. Rules are applied in order. The first rule that matches applies, and the permission in each rule overrides the permissions in all following rules. Use the **Move Up** and **Move Down** buttons to change the order of the selected rule.

Rules that appear dimmed are inherited from the parent and cannot be changed at this level.

Permission	Users and Roles	Delete
Allow	Administrator	Delete
Allow	Human Resources	Delete
Deny	Sales	Delete
Deny	[all]	Delete
Allow	[all]	Delete

Figure 33-14

are used to manually modifying this tag in previous versions of ASP.NET. Figure 33-15 shows the Application tab.

Clicking the Create Application Settings link takes you to a screen where you can provide the name and the value information. Clicking Manage Application Settings takes you to a screen where you can view existing settings and edit or delete them. You can also create new setting from this screen.

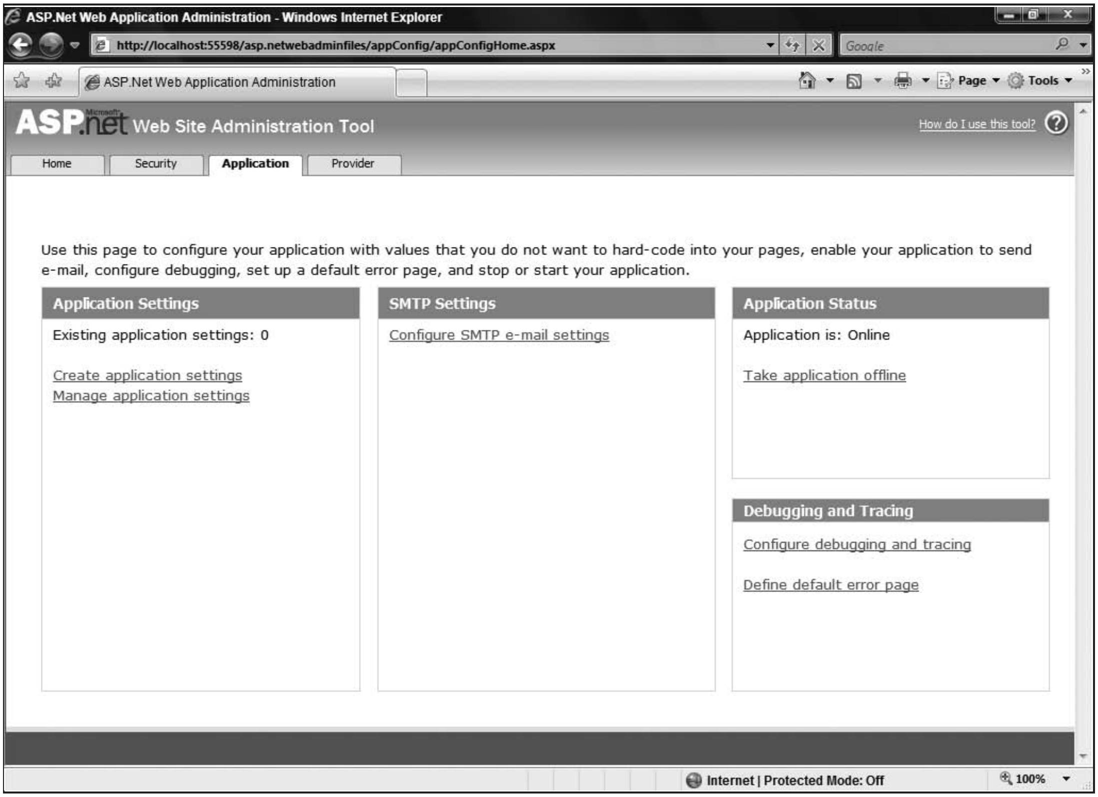
Managing SMTP Configuration

Click the Configure SMTP E-Mail Settings link to view a screen like the one shown in Figure 33-16. The configure SMTP mail settings feature is useful if your Web application can send autogenerated e-mails. Instead of denoting SMTP server configuration in the code, you can spell it out in the configuration file by entering values here in the administration tool.

Specify the server name, port, sender e-mail address, and authentication type.

Managing Tracing and Debugging Information

Clicking the Application tab's Configure Debugging and Tracing link takes you to a screen (see Figure 33-17) where you can enable or disable tracing and debugging. Select whether you want to display trace information on each page. You can also specify whether to track just local requests or all requests, as well as trace sorting and caching configuration.


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/appConfig/appConfigHome.aspx
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Configure SMTP e-mail settings
Take application offline
Configure debugging and tracing
Define default error page
Internet | Protected Mode: Off
100%

Use this page to configure your application with values that you do not want to hard-code into your pages, enable your application to send e-mail, configure debugging, set up a default error page, and stop or start your application.

Application Settings

Existing application settings: 0

Create application settings
Manage application settings

SMTP Settings

Application Status

Application is: Online

Debugging and Tracing

Figure 33-15

To configure default error pages, you simply click Define Default Error Page on the screen you saw in Figure 33-15. This takes you to a screen where you can select a URL that is used for redirection in case of an error condition (see Figure 33-18).

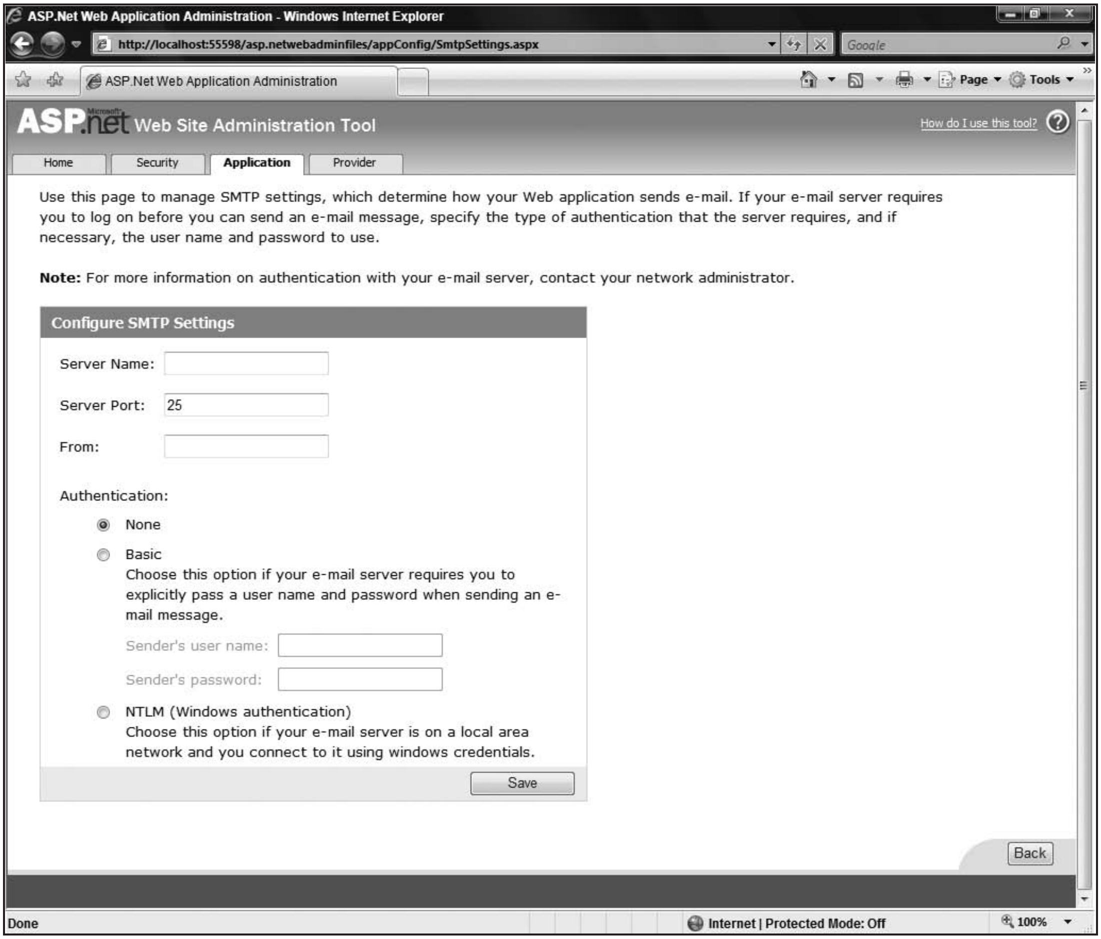
Taking an Application Offline

You can take your entire Web application offline simply by clicking the Take Application Offline link (again, refer to Figure 33-15). The link stops the app domain for your Web application. It is useful if you want to perform a scheduled maintenance for an application.

The Provider Tab

The final tab in the ASP.NET Web Site Administration Tool is Provider, shown in Figure 33-19. You use it to set up additional providers and to determine the providers your application will use.

The Provider page is simple, but it contains an important piece of information: the default data provider with which your application is geared to work. In Figure 33-19, the application is set up to work with the `AspNetSqlProvider` provider, the default data provider.


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/appConfig/SmtpSettings.aspx
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Save
Back
Done
Internet | Protected Mode: Off
100%

Use this page to manage SMTP settings, which determine how your Web application sends e-mail. If your e-mail server requires you to log on before you can send an e-mail message, specify the type of authentication that the server requires, and if necessary, the user name and password to use.

Note: For more information on authentication with your e-mail server, contact your network administrator.

Configure SMTP Settings

Server Name:

Server Port: 25

From:

Authentication:

● None

○ Basic
Choose this option if your e-mail server requires you to explicitly pass a user name and password when sending an e-mail message.

Sender's user name:

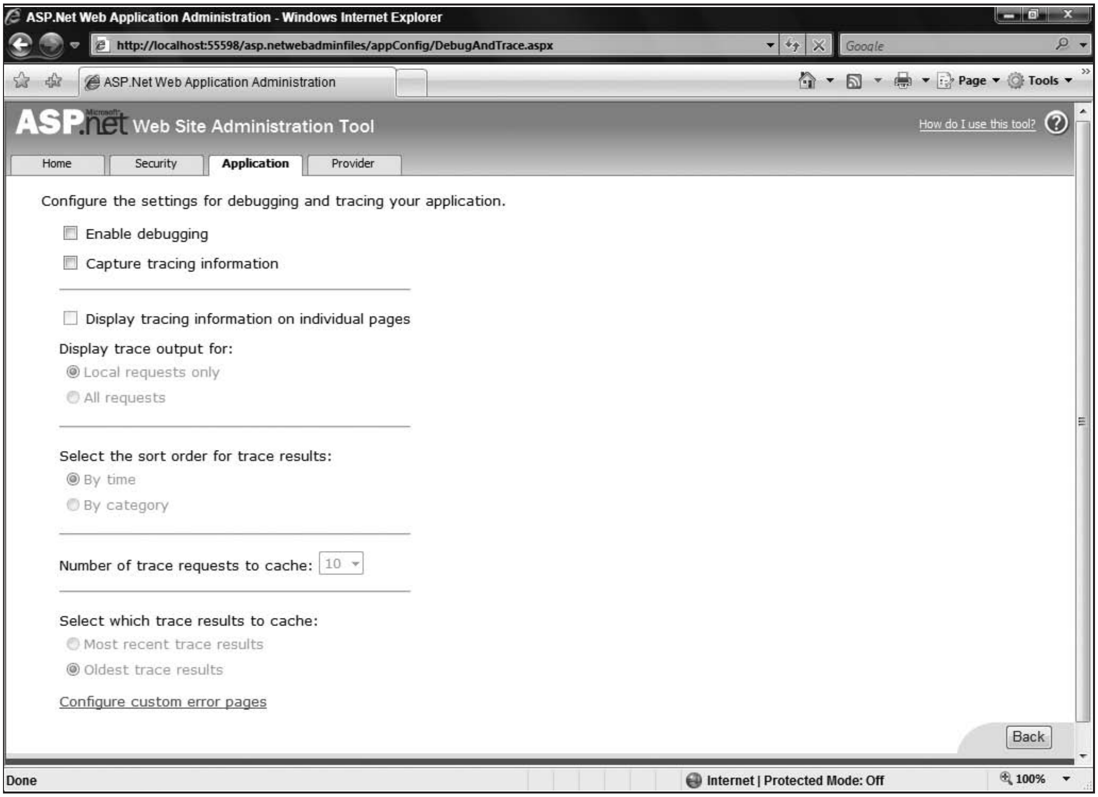
Sender's password:

○ NTLM (Windows authentication)
Choose this option if your e-mail server is on a local area network and you connect to it using windows credentials.

Figure 33-16

The two links on this tab let you set up either a single data provider or a specific data provider for each of the features in ASP.NET that requires a data provider. If you click the latter, you are presented with the screen shown in Figure 33-20. It enables you to pick the available providers separately for Membership and Role management.

As you can see from the screenshots and brief explanations provided here, you could now handle a large portion of the necessary configurations through a GUI. You no longer have to figure out which setting must be placed in the `web.config` file. This functionality becomes even more important as the `web.config` file grows. In ASP.NET 1.0/1.1, the `web.config` file was a reasonable size, but with all the features provided by ASP.NET 2.0 or 3.5, the `web.config` file has the potential to become very large. These GUI-based tools are an outstanding way to configure some of the most commonly needed settings. However, there are many settings that can not be modified with the Web Server Administration Tool, such as the AJAX settings, so you will still need to edit web.config in many cases.


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/appConfig/DebugAndTrace.aspx
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Back
Done
Internet | Protected Mode: Off
100%

Configure the settings for debugging and tracing your application.

☐ Enable debugging

☐ Capture tracing information

☐ Display tracing information on individual pages

Display trace output for:

● Local requests only

○ All requests

Select the sort order for trace results:

● By time

○ By category

Number of trace requests to cache: 10

Select which trace results to cache:

○ Most recent trace results

● Oldest trace results

Configure custom error pages

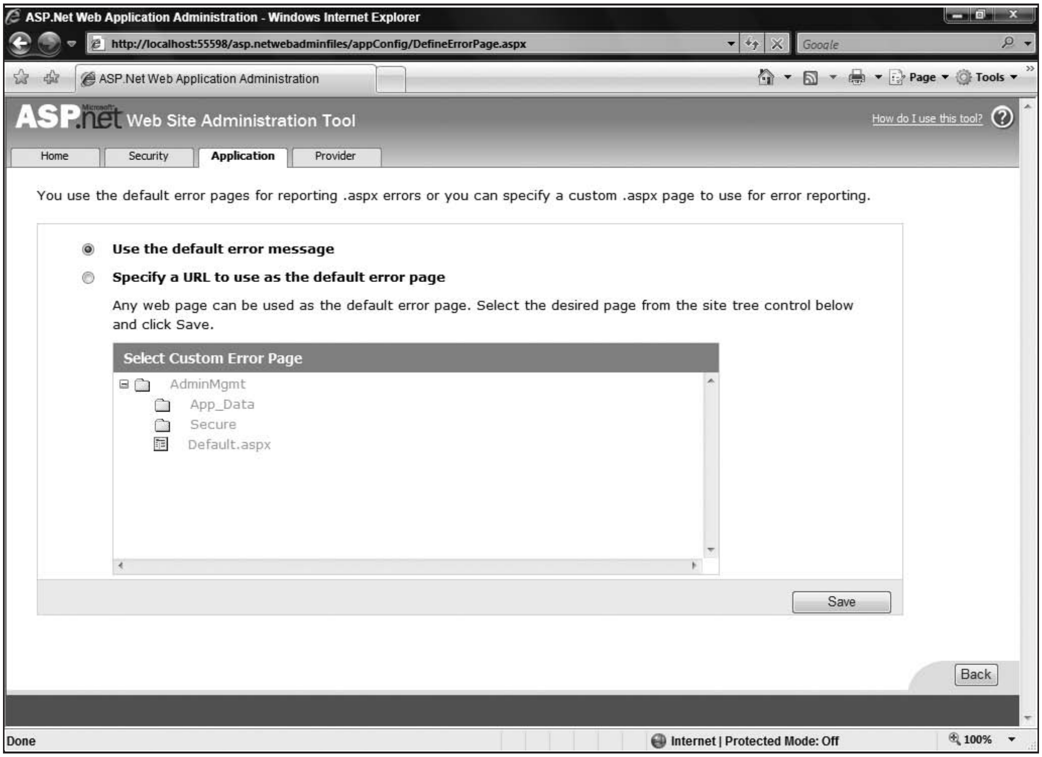
Figure 33-17

Configuring ASP.NET in IIS on Vista

If you are using IIS as the basis of your ASP.NET applications, you will find that it is quite easy to configure the ASP.NET application directly through the Internet Information Services (IIS) Manager if you are using Windows Vista. To access the ASP.NET configurations, open IIS and expand the `Web Sites` folder, which contains all the sites configured to work with IIS. Remember that not all your Web sites are configured to work in this manner because it is also possible to create ASP.NET applications that make use of the new ASP.NET built-in Web server.

Once you have expanded the IIS Web Sites folder, right-click one of the applications in this folder and you will notice that the options you have available to you for configuration will appear in the IIS Manager (see Figure 33-21).

The options available to you enable you to completely configure ASP.NET or even configure IIS itself. The focus of this chapter is on the ASP.NET section of the options. In addition to the options you can select from one of the available icons, you can also configure some basic settings of the application by clicking the Basic Settings link in the Actions pane on the right-hand side of the IIS Manager. When clicking the Basic Settings link, you will get a dialog box, as shown in Figure 33-22.


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/appConfig/DefineErrorPage.aspx
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider

You use the default error pages for reporting .aspx errors or you can specify a custom .aspx page to use for error reporting.

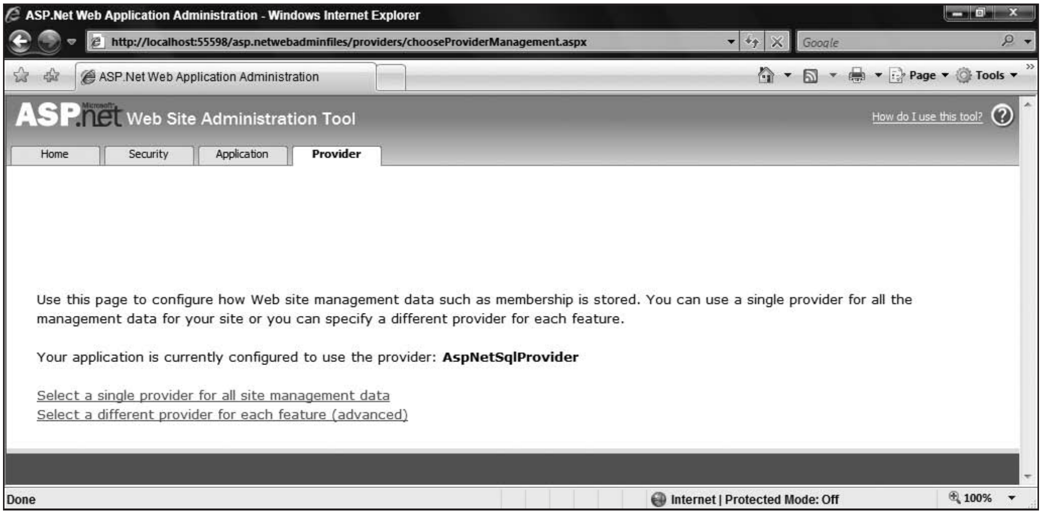
● **Use the default error message**

○ **Specify a URL to use as the default error page**

Any web page can be used as the default error page. Select the desired page from the site tree control below and click Save.

Select Custom Error Page
AdminMgmt
App_Data
Secure
Default.aspx
Save
Back
Done
Internet | Protected Mode: Off
100%

Figure 33-18


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:55598/asp.netwebadminfiles/providers/chooseProviderManagement.aspx
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider

Use this page to configure how Web site management data such as membership is stored. You can use a single provider for all the management data for your site or you can specify a different provider for each feature.

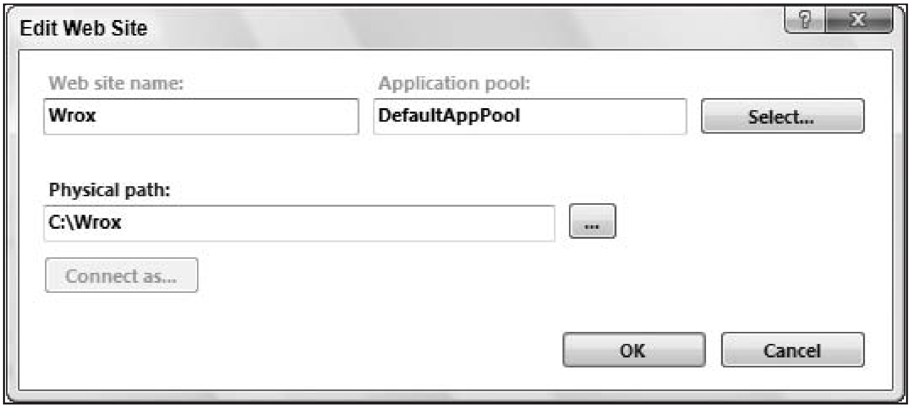
Your application is currently configured to use the provider: **AspNetSqlProvider**

Select a single provider for all site management data
Select a different provider for each feature (advanced)

Done
Internet | Protected Mode: Off
100%

Figure 33-19

Changes you are making in the IIS Manager are actually being applied to the `web.config` *file of your application; making changes to the Default Web site (the root node) lets you edit the* `machine.config` *file.*

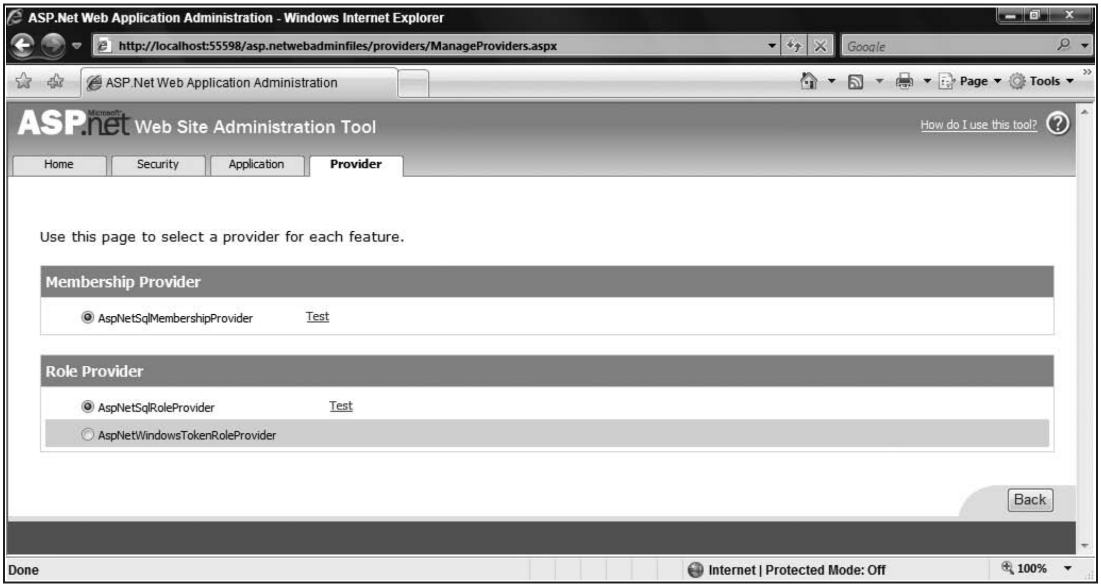

Edit Web Site
Select...
...
Connect as...
OK
Cancel

Web site name: Wrox

Application pool: DefaultAppPool

Physical path: C:\Wrox

Figure 33-20


ASP.Net Web Application Administration - Windows Internet Explorer
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Back
Done
Internet | Protected Mode: Off
100%

http://localhost:55598/asp.netwebadminfiles/providers/ManageProviders.aspx

Use this page to select a provider for each feature.

Membership Provider

● AspNetSqlMembershipProvider Test

Role Provider

● AspNetSqlRoleProvider Test

○ AspNetWindowsTokenRoleProvider

Figure 33-21

The dialog enables you to change the following items:

- **Web site name:** The name of the Web site. In the case of Figure 33-22, naming the Web site "Wrox" means that the URL will be `http://[IP address or domain name]/Wrox`.

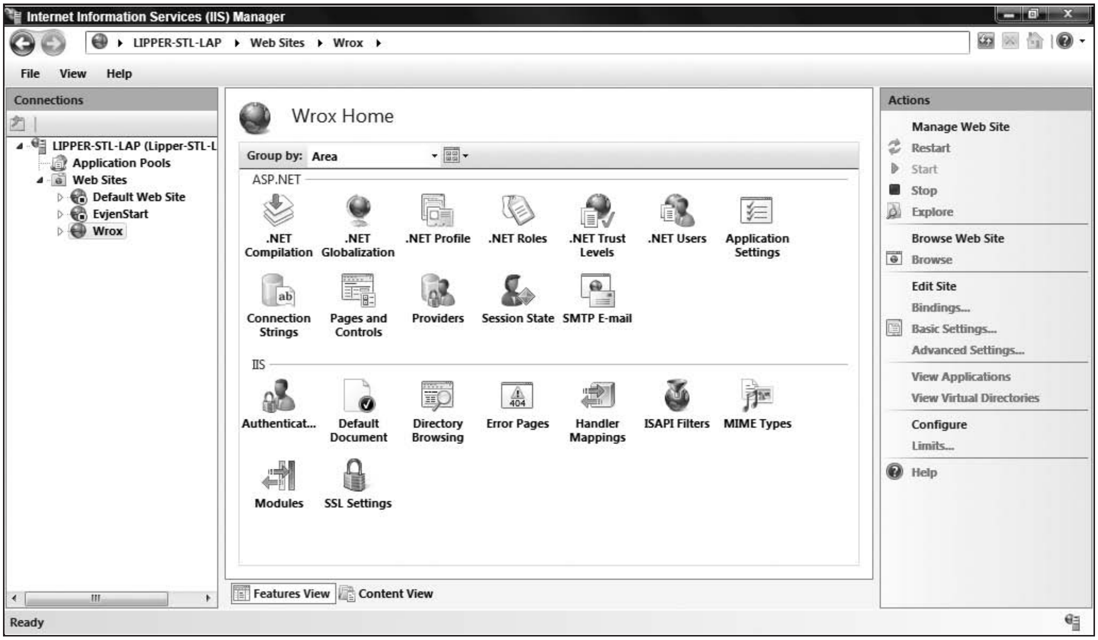

Internet Information Services (IIS) Manager
LIPPER-STL-LAP ▸ Web Sites ▸ Wrox ▸
File
View
Help
Connections
LIPPER-STL-LAP (Lipper-STL-L
Application Pools
Web Sites
Default Web Site
EvjenStart
Wrox
Wrox Home
Group by: Area
ASP.NET
.NET Compilation
.NET Globalization
.NET Profile
.NET Roles
.NET Trust Levels
.NET Users
Application Settings
Connection Strings
Pages and Controls
Providers
Session State
SMTP E-mail
IIS
Authenticat...
Default Document
Directory Browsing
Error Pages
Handler Mappings
ISAPI Filters
MIME Types
Modules
SSL Settings
Features View
Content View
Actions
Manage Web Site
Restart
Start
Stop
Explore
Browse Web Site
Browse
Edit Site
Bindings...
Basic Settings...
Advanced Settings...
View Applications
View Virtual Directories
Configure
Limits...
Help
Ready

Figure 33-22

- **Application pool:** The application pool you are going to use for the application. You will notice that you have three options by default — DefaultAppPool (which uses the .NET Framework 2.0 and an integrated pipeline mode), Classic .NET AppPool (which uses the .NET Framework 2.0 and a classic pipeline mode), and ASP.NET 1.1 (which uses the .NET Framework 1.1 as it states and a classic pipeline mode).
- **Physical path:** The folder location where the ASP.NET application can be found. In this case, it is `C:\Wrox`.

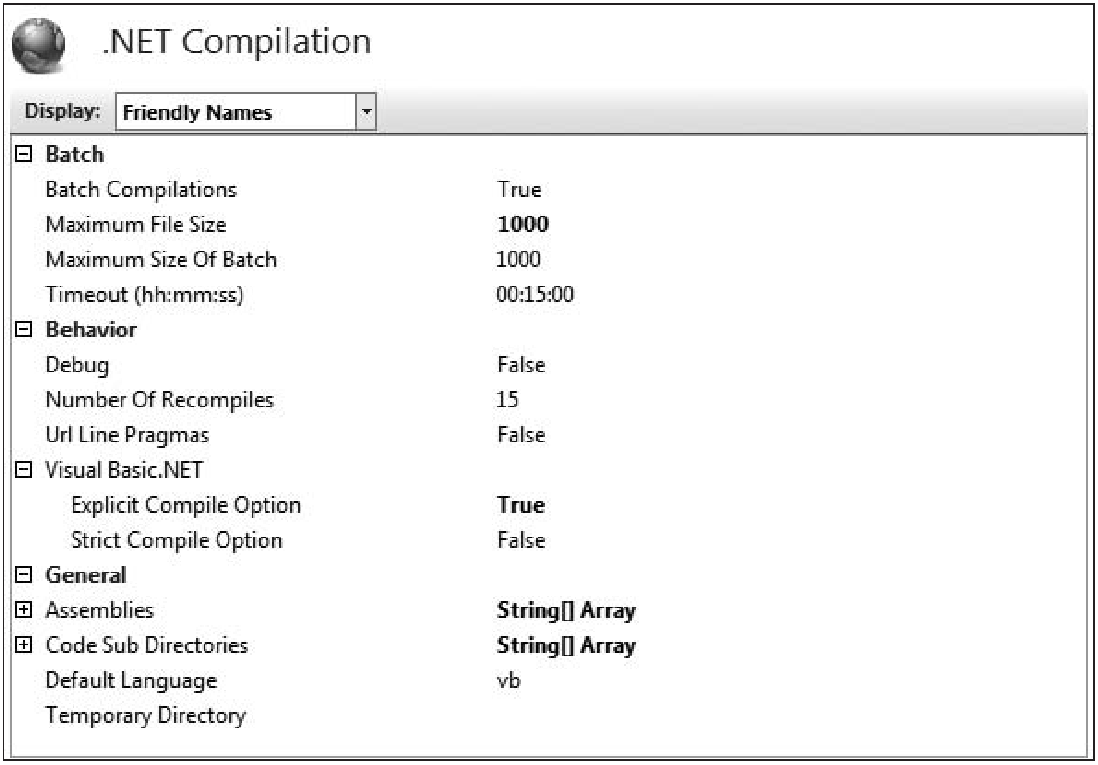
The sections that follow review some of the options available to you through the icons in the IIS Manager.

.NET Compilation

Use the Application tab to make changes that are more specific to the pages in the context of your application. From this dialog, shown in Figure 33-23, you can change how your pages are compiled and run. You can also make changes to global settings in your application.

This section deals with compilation of the ASP.NET application as well as how some of the pages of the application will behave. The Batch section deals with the batch compilation of the application — first, whether or not it is even supported, and then details on batch sizes and the time it takes to incur the compilation.

The Behavior section deals with whether or not the compilation produces a release or debug build; you will also find some Visual Basic–specific compilation instructions on whether Option Explicit or Option Script are enabled across the entire application.



.NET Compilation

Display: Friendly Names

⊟ **Batch**	
Batch Compilations	True
Maximum File Size	**1000**
Maximum Size Of Batch	1000
Timeout (hh:mm:ss)	00:15:00
⊟ **Behavior**	
Debug	False
Number Of Recompiles	15
Url Line Pragmas	False
⊟ Visual Basic.NET	
Explicit Compile Option	**True**
Strict Compile Option	False
⊟ **General**	
⊞ Assemblies	**String[] Array**
⊞ Code Sub Directories	**String[] Array**
Default Language	vb
Temporary Directory	

Figure 33-23

The General section focuses on the assemblies that are referenced as well as your code subdirectories if you are going to break up your App_Code folder into separate compiled instances (required for when you want to incorporate Visual Basic and C# code in the same application). You can also specify the default language that is used in the compilation process — such as `VB` or `C#`.

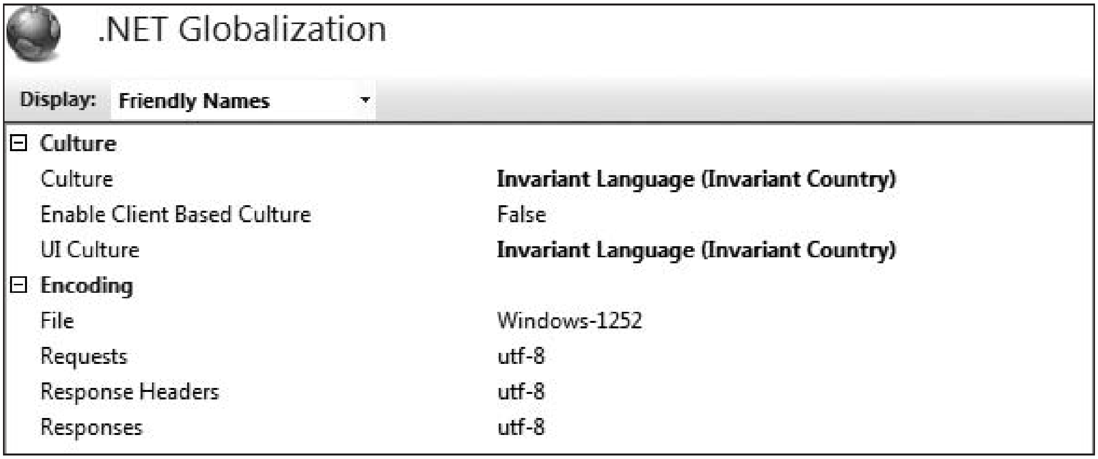
.NET Globalization

The .NET Globalization option enables you to customize how your ASP.NET application deals with culture and the encoding of the requests and responses. Figure 33-24 shows the options available in this section.

In addition to picking a specific Culture or UI Culture setting, you can also select Auto Detect, which will pick up the culture of the client if it is available. By default, you can also see that the encoding of the requests and the responses are set to `utf-8`, which will work fine for most Latin-based languages.

.NET Profile

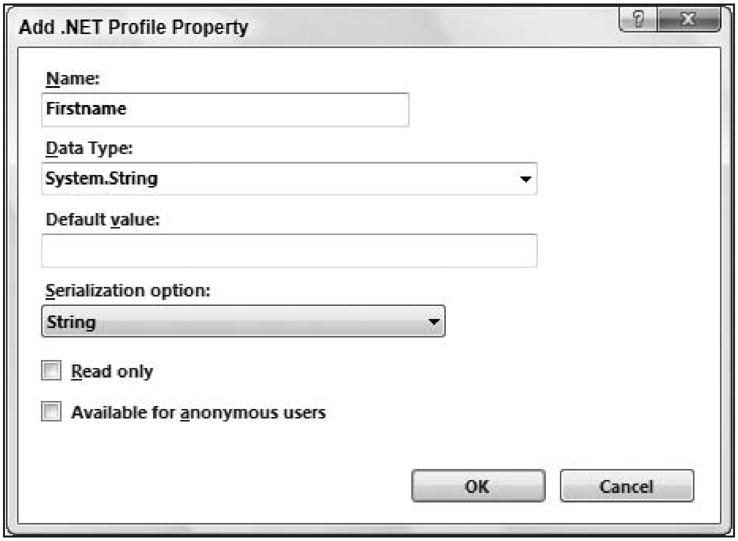
The .NET Profile options enable you to customize how your ASP.NET application deals with the ASP.NET personalization system. This system was discussed earlier in Chapter 15 of this book. Figure 33-25 shows the dialog that is provided when you add a new profile to the personalization system.



.NET Globalization

Culture	
Culture	**Invariant Language (Invariant Country)**
Enable Client Based Culture	False
UI Culture	**Invariant Language (Invariant Country)**
Encoding	
File	Windows-1252
Requests	utf-8
Response Headers	utf-8
Responses	utf-8

Figure 33-24



Add .NET Profile Property

Name: Firstname

Data Type: System.String

Default value:

Serialization option: String

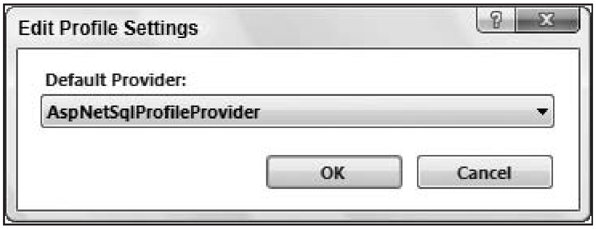
☐ Read only

☐ Available for anonymous users

Figure 33-25

In this case, as presented in Figure 33-25, you can specify the name of the personalization property, the data type used, its default value, how it is serialized, and whether or not it is read-only or available for anonymous users. To better understand these settings, it is important to review Chapter 15.

In addition to building properties to use in the personalization system, you can also specify the provider that is used by the system as a whole. By default, it will be using the AspNetSqlProfileProvider, as illustrated in Figure 33-26.

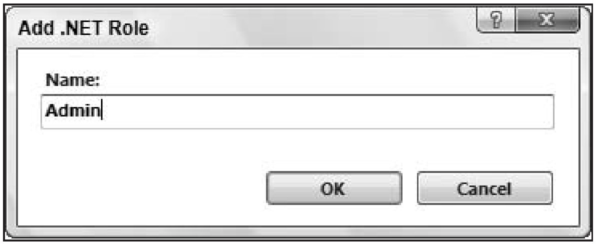


Default Provider: AspNetSqlProfileProvider

Figure 33-26

.NET Roles

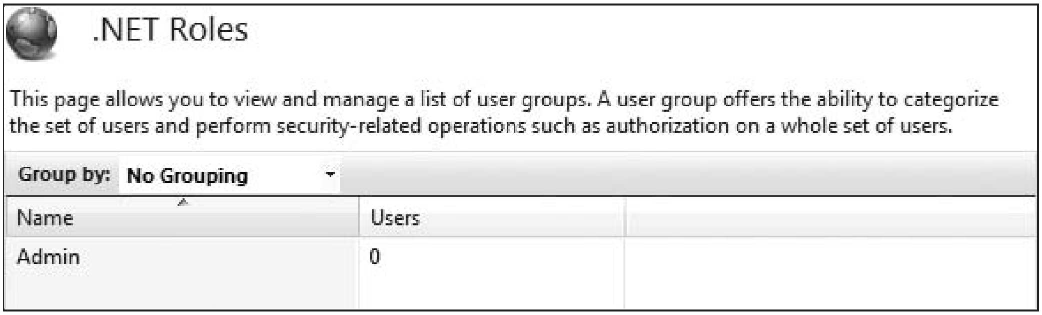
You can enable role-based management by adding roles to your application from the .NET Roles section. Figure 33-27 shows an example of adding a role called Admin to the application.



Name: Admin

Figure 33-27

Pressing OK will add the role to the system and the role will then be shown in a list of roles from the main screen of the section, as illustrated in Figure 33-28.



This page allows you to view and manage a list of user groups. A user group offers the ability to categorize the set of users and perform security-related operations such as authorization on a whole set of users.

Group by: No Grouping

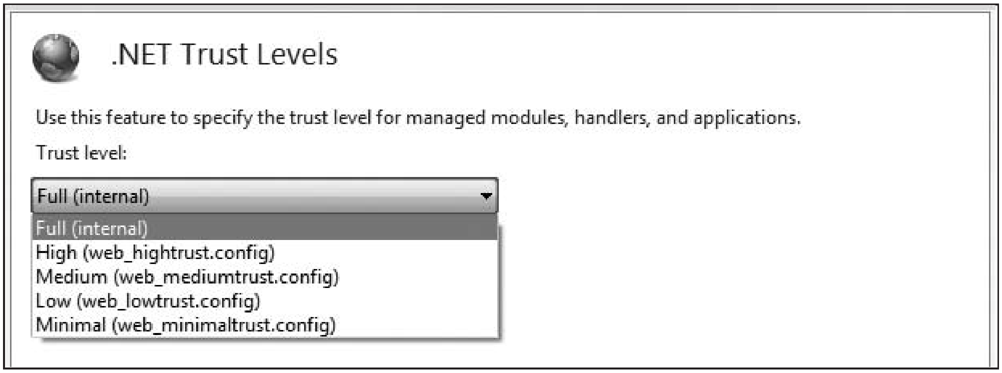
Name	Users
Admin	0

Figure 33-28

By default, there will be no users added to the role. You will be able to add users to roles through the .NET Users section, discussed shortly.

.NET Trust Levels

The .NET Trust Levels section allows you to specify the level of security to apply to your application through the selection of a specific pre-generated configuration file. This is illustrated in the list of options presented in Figure 33-29.



.NET Trust Levels

Use this feature to specify the trust level for managed modules, handlers, and applications.

Trust level:

Full (internal)

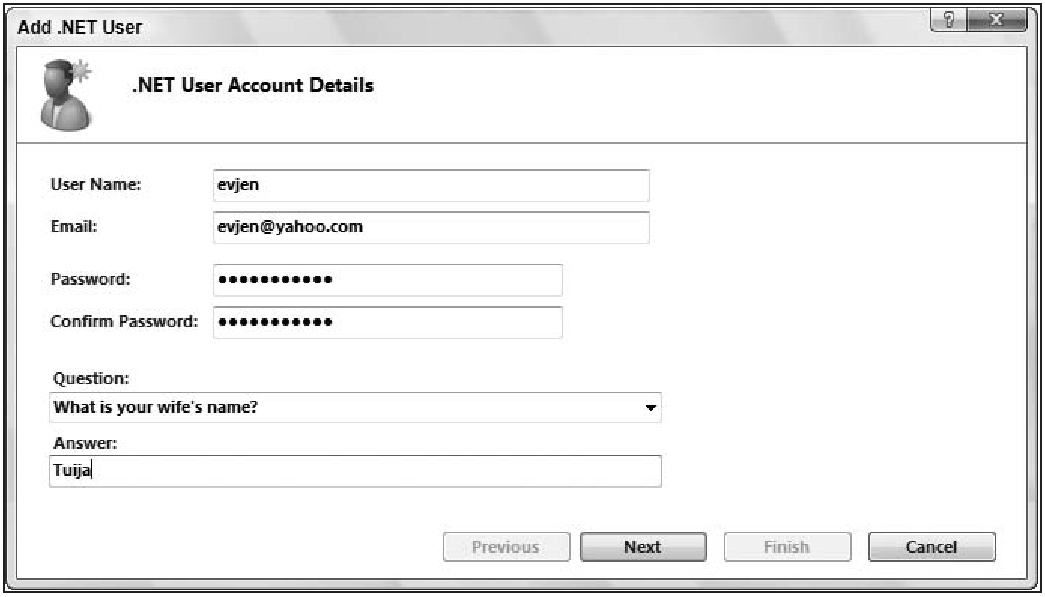
Full (internal)
High (web_hightrust.config)
Medium (web_mediumtrust.config)
Low (web_lowtrust.config)
Minimal (web_minimaltrust.config)

Figure 33-29

By default, your application makes use of the `web.config` file, but specifying a different trust level will cause the application to use a different `.config` file. All of these `.config` files are found at `C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG`.

.NET Users

Probably one of the easiest ways to work with the ASP.NET membership system (covered in Chapter 16 of this book) is to create your users in the .NET Users section of IIS. Adding a user is easy through the dialogs provided, as illustrated in Figure 33-30.



Add .NET User

.NET User Account Details

User Name: evjen

Email: evjen@yahoo.com

Password: ●●●●●●●●●●●●

Confirm Password: ●●●●●●●●●●●●

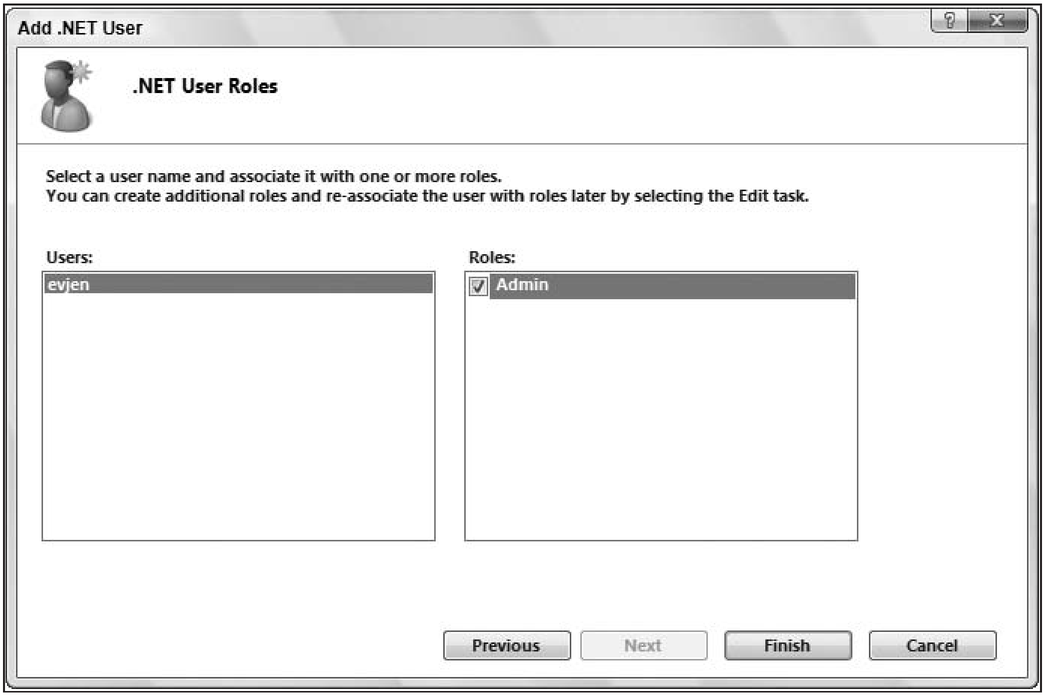
Question: What is your wife's name?

Answer: Tuija

Previous
Next
Finish
Cancel

Figure 33-30

In Figure 33-30, you can provide the username, password, and security question and answer in a simple wizard. Figure 33-31 shows the second screen of the wizard.



Select a user name and associate it with one or more roles.
You can create additional roles and re-associate the user with roles later by selecting the Edit task.

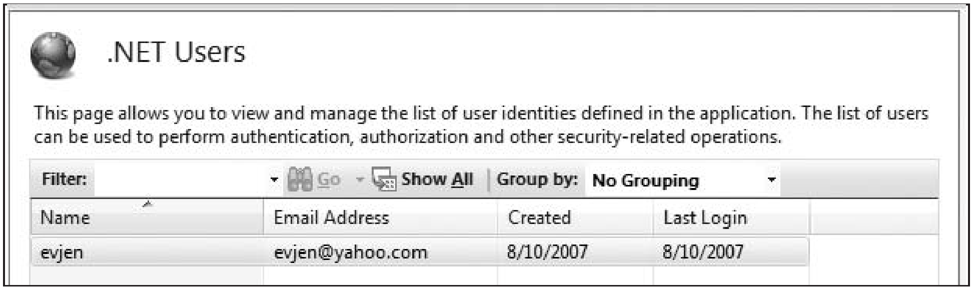
Users: evjen

Roles: ☑ Admin

Figure 33-31

In this second screen of the wizard, you can assign users to specific roles that are present in the role management system. Because the Admin role was created earlier in this chapter, I am able to assign the user to this particular role as it exists in the system.

Once a user is created, you can then see the entire list of users for this particular application from the main .NET Users screen, as illustrated in Figure 33-32.



This page allows you to view and manage the list of user identities defined in the application. The list of users can be used to perform authentication, authorization and other security-related operations.

Name	Email Address	Created	Last Login
evjen	evjen@yahoo.com	8/10/2007	8/10/2007

Figure 33-32

Application Settings

Another section is the Application Settings section. Click its Add or Edit button, and the Edit/Add Application Settings dialog opens (see Figure 33-33).

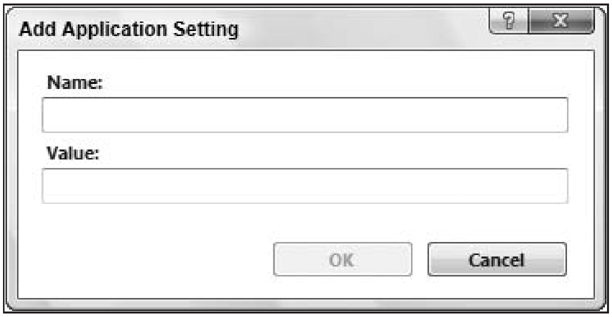
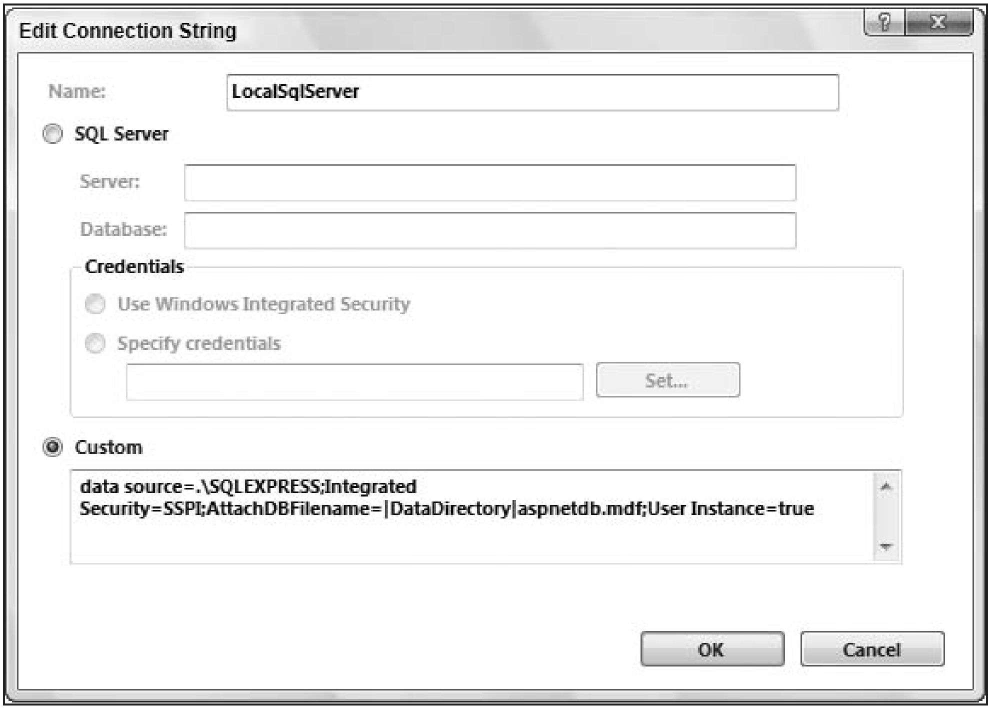

Add Application Setting
Name:
Value:
OK
Cancel

Figure 33-33

After you enter a key and value pair, click OK; the settings appear in the list in the main dialog. Then you can edit or delete the settings from the application.

Connection Strings

The next section is the Connection Strings section. To add a connection string to your application, just click its Add button. You also can edit or remove existing connection strings. Figure 33-34 shows the Edit Connection String dialog for the default connection string — LocalSqlServer.


Edit Connection String
SQL Server
Server:
Database:
Credentials
Use Windows Integrated Security
Specify credentials
Set...
OK
Cancel

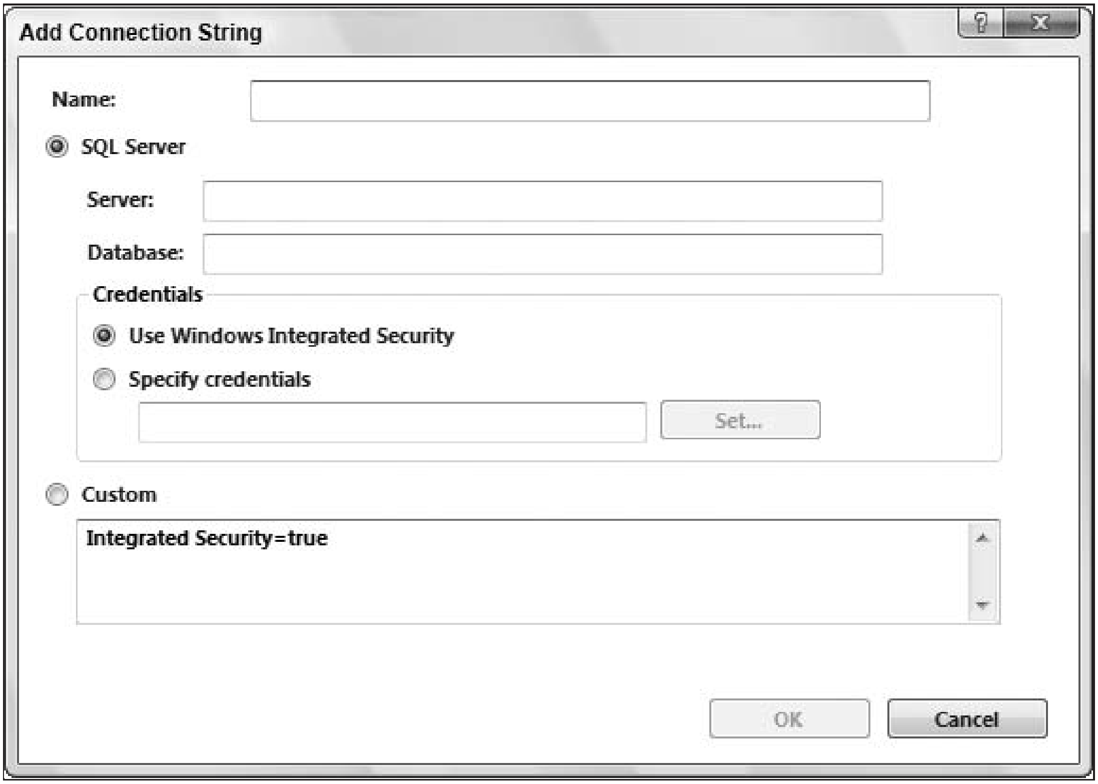
Name: LocalSqlServer

● Custom

```
data source=.\SQLEXPRESS;Integrated
Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;User Instance=true
```

Figure 33-34

It is also rather simple to add a brand new connection, as illustrated in Figure 33-35.



Add Connection String

Name:

● SQL Server

Server:

Database:

Credentials

● Use Windows Integrated Security

○ Specify credentials

○ Custom

Integrated Security=true

Figure 33-35

Pages and Controls

The Pages and Controls section deals with a group of settings that control the overall ASP.NET pages (`.aspx`) and user controls in the application (`.ascx`). Figure 33-36 shows the available settings for this section.

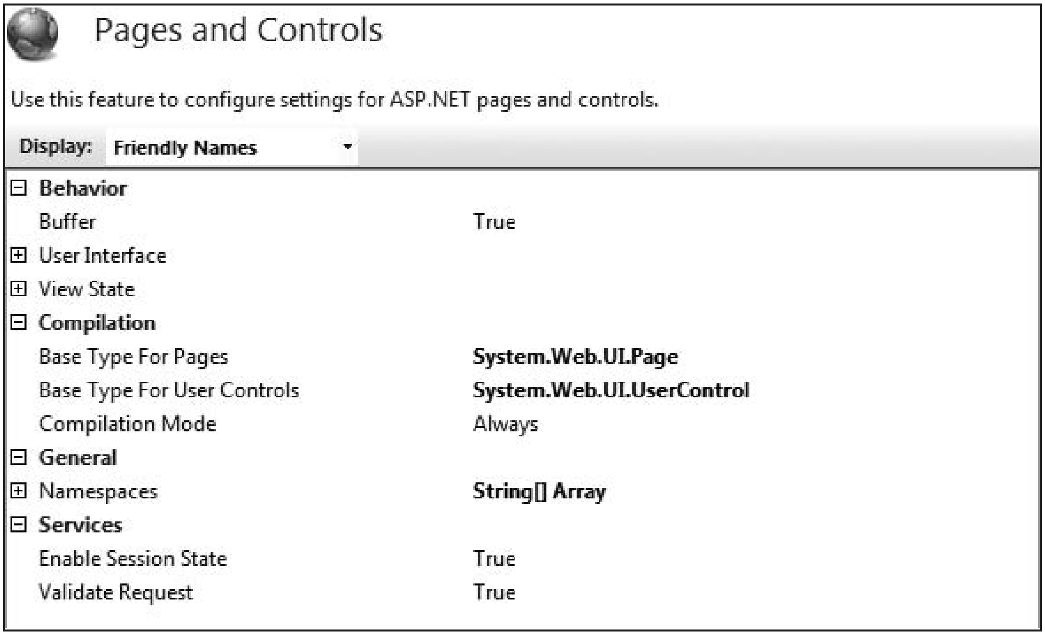
Providers

The Providers section deals with all the providers that are defined within the application. From the example in Figure 33-37, you can see that there are only two providers defined for the .NET Roles engine — a SQL Server role provider and a Windows Token role provider.

You can look at all the other engines found in ASP.NET by selecting the option in the drop-down list at the top of the dialog.

Session State

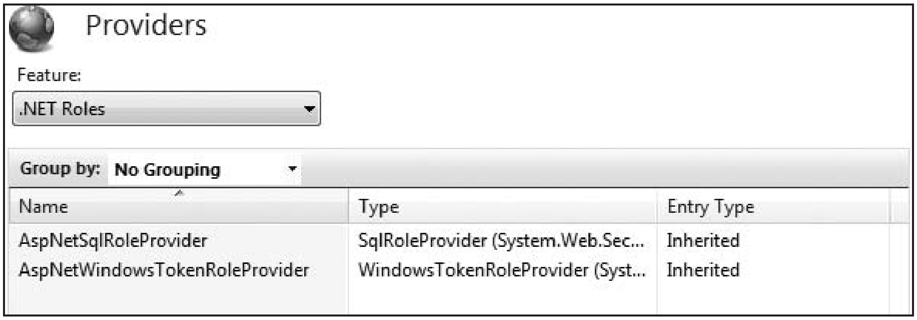
ASP.NET applications, being stateless in nature, are highly dependent on how state is stored. The Session State section (see Figure 33-38) enables you to change a number of different settings that determine how state management is administered.



Pages and Controls
Use this feature to configure settings for ASP.NET pages and controls.
Display: Friendly Names

⊟ **Behavior**	
Buffer	True
⊞ User Interface	
⊞ View State	
⊟ **Compilation**	
Base Type For Pages	**System.Web.UI.Page**
Base Type For User Controls	**System.Web.UI.UserControl**
Compilation Mode	Always
⊟ **General**	
⊞ Namespaces	**String[] Array**
⊟ **Services**	
Enable Session State	True
Validate Request	True

Figure 33-36

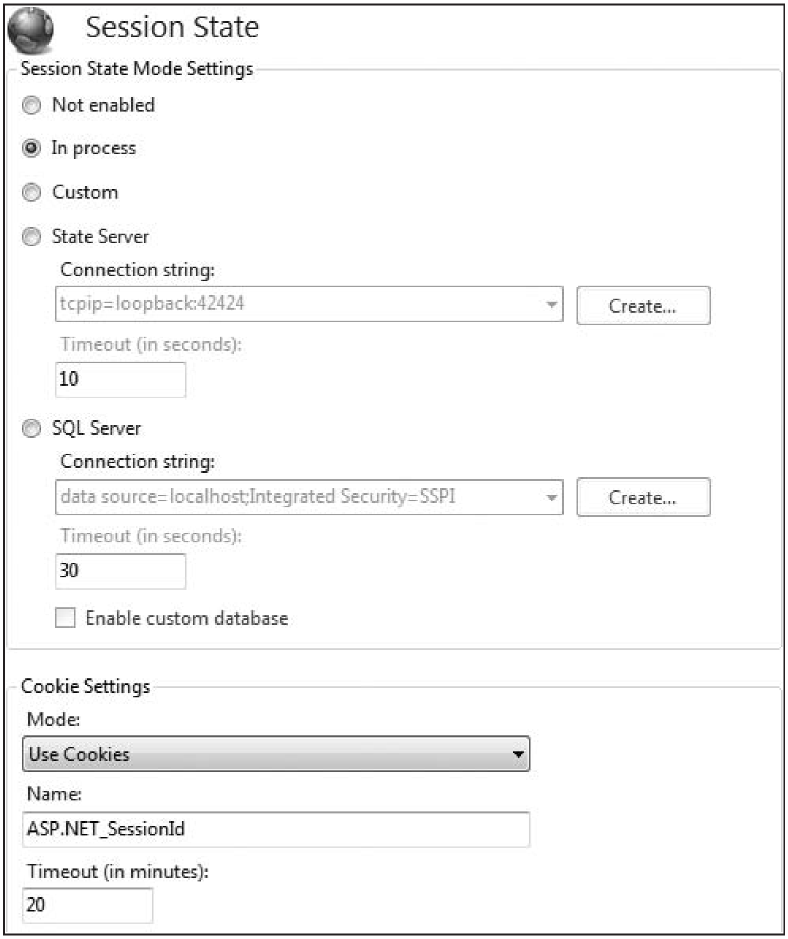


Name	Type	Entry Type
AspNetSqlRoleProvider	SqlRoleProvider (System.Web.Sec...	Inherited
AspNetWindowsTokenRoleProvider	WindowsTokenRoleProvider (Syst...	Inherited

Figure 33-37

You can apply state management to your applications in a number of ways, and this dialog allows for a number of different settings — some of which are enabled or disabled based on what is selected. The following list describes the items available in the Session State Settings section:

- **Session state mode:** Determines how the sessions are stored by the ASP.NET application. The default option (shown in Figure 33-38) is InProc. Other options include Off, StateServer, and SQLServer. Running sessions in-process (InProc) means that the sessions are stored in the same process as the ASP.NET worker process. Therefore, if IIS is shut down and then brought up again, all the sessions are destroyed and unavailable to end users. StateServer means that sessions are stored out-of-process by a Windows service called ASPState. SQLServer is by far the most secure way to deal with your sessions — it stores them directly in SQL Server. StateServer is also the least performance-efficient method.



Session State

Session State Mode Settings

○ Not enabled
● In process
○ Custom
○ State Server
  Connection string: tcpip=loopback:42424
  Timeout (in seconds): 10
○ SQL Server
  Connection string: data source=localhost;Integrated Security=SSPI
  Timeout (in seconds): 30
  ☐ Enable custom database

Cookie Settings

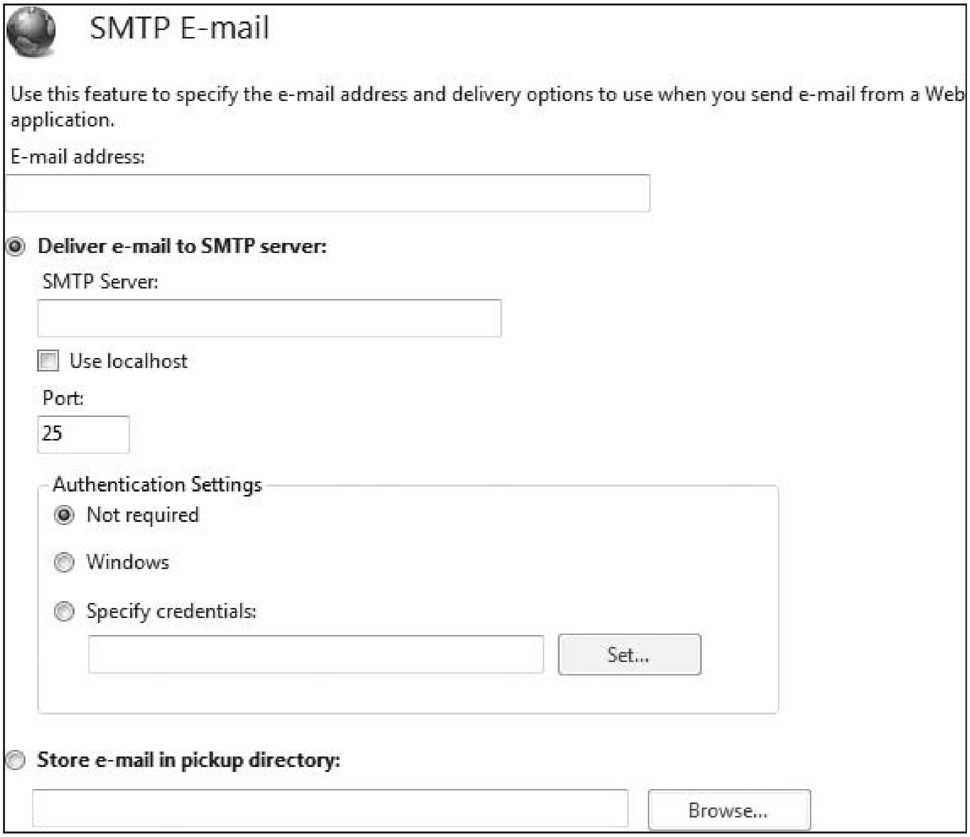
Mode: Use Cookies
Name: ASP.NET_SessionId
Timeout (in minutes): 20

Figure 33-38

- **Cookieless mode:** Changes how the identifiers for the end user are stored. The default setting uses cookies (UseCookies). Other possible settings include UseUri, AutoDetect, and UseDeviceProfile.
- **Session timeout:** Sessions are stored for only a short period of time before they expire. For years, the default has been 20 minutes. Modifying the value here changes how long the sessions created by your application are valid.

SMTP E-mail

If you need to work with an application that delivers e-mail, then you must specify the settings to do this. Sending e-mail and the settings required are defined in the SMTP E-mail section (see Figure 33-39).



Use this feature to specify the e-mail address and delivery options to use when you send e-mail from a Web application.

E-mail address:

● Deliver e-mail to SMTP server:

SMTP Server:

☐ Use localhost

Port: 25

● Not required

○ Windows

○ Specify credentials:

○ Store e-mail in pickup directory:

Figure 33-39

Summary

This chapter showed you some of the new management tools that come with the latest release of ASP.NET. These tools make the ever-increasing size of the `web.config` file more manageable because they take care of setting the appropriate values in the application's configuration file.

The new IIS Manager console in Windows Vista is a welcome addition for managing applications that are configured to work with IIS. The ASP.NET Web Site Administration Tool provides even more value to administrators and developers by enabling them to easily manage settings.

34

Packaging and Deploying ASP.NET Applications

Packaging and deploying ASP.NET applications are topics that usually receive little attention. This chapter is going to take a more in-depth look at how you can package and deploy your ASP.NET applications after they are built. After you have developed your ASP.NET application on a development computer, you will need to deploy the finished product to a quality assurance or staging server, and eventually on a production server.

An important reason to consider the proper packaging and deploying of your ASP.NET applications is that many applications are built as saleable products, starter kits, or solutions. In this case you may have to allow complete strangers to download and install these products in their own — environments that you have absolutely no control over. If this is the case, it is ideal to give the consumer a single installer file that ensures proper installation of the application in any environment.

But regardless of whether you will distribute your web application outside your company you still need a way to deploy it to another server where it can be tested before production deployment. You should never assume that it will be perfect just because it worked on your computer. Most of the time we just develop using the internal web server in Visual Studio, so we will need a full test using IIS before we assume all is well. Even if you do test with IIS on your computer there are still deployment related factors that need to be ironed out and fully tested before the application goes to production.

Before you start, you should understand the basics of packaging and deploying ASP.NET applications. In the process of packaging your ASP.NET applications, you are putting your applications into a package and utilizing a process of deployment that is initiated through a deployment procedure, such as using a Windows installer.

Deployment Pieces

So what are you actually deploying? ASP.NET contains a lot of pieces that are all possible parts of the overall application and need to be deployed with the application in order for it to run properly. The following list details some of the items that are potentially part of your ASP.NET application and need deployment consideration when you are moving your application:

- `.aspx` pages
- The code-behind pages for the `.aspx` pages (`.aspx.vb` or `.aspx.cs` files)
- User controls (`.ascx`)
- Web service files (`.asmx` and `.wsdl` files)
- `.htm` or `.html` files
- Image files such as `.jpg` or `.gif`
- ASP.NET system folders such as `App_Code` and `App_Themes`
- JavaScript files (`.js`)
- Cascading Style Sheets (`.css`)
- Configuration files such as the `web.config` file
- .NET components and compiled assemblies
- Data files such as `.mdb` files

Steps to Take before Deploying

Before deploying your ASP.NET Web applications, you should take some basic steps to ensure that your application is *ready* for deployment. These steps are often forgotten and are mentioned here to remind you of how you can ensure that your deployed application performs at its best.

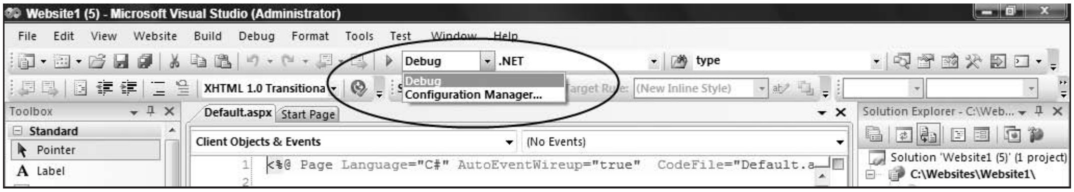
The first step you should take is to *turn off* debugging in the `web.config` file. You do this by setting the `debug` attribute in the `<compilation>` element to `false`, as shown in Listing 34-1.

Listing 34-1: Setting debug to false before application deployment

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
   <system.web>

      <compilation debug="false" />

   </system.web>
</configuration>
```

By default, most developers set the `debug` attribute to `true` when developing their applications. Doing this inserts debug symbols into the compiled ASP.NET pages. These symbols degrade the performance of any application. After the application is built and ready to be deployed, it is unnecessary to keep these debug symbols in place.

For those that have been coding ASP.NET for some time now, it is important to note that the Debug option in the drop-down list in the Visual Studio menu does not accomplish much in changing the configuration file or anything similar (shown here in Figure 34-1). In the ASP.NET 1.0 and 1.1 days, Visual Studio .NET (as it was called at that time) actually controlled the compilation of the ASP.NET project to a DLL. Now, and ever since ASP.NET 2.0, it is actually ASP.NET itself that controls the compilation process at runtime. Therefore, while the drop-down with the Debug designation is present, it really has no meaning in the context of building an ASP.NET project. You completely control the compilation designation through what is set in the `web.config` file, as shown in Listing 34-1.


Website1 (5) - Microsoft Visual Studio (Administrator)
File
Edit
View
Website
Build
Debug
Format
Tools
Test
Window
Help
Debug
.NET
type
Debug
Configuration Manager...
XHTML 1.0 Transitiona
Target Rule:
(New Inline Style)
Toolbox
Standard
Pointer
Label
Default.aspx
Start Page
Client Objects & Events
(No Events)
Solution Explorer - C:\Web...
Solution 'Website1 (5)' (1 project)
C:\Websites\Website1\

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.a
```

Figure 34-1

Methods of Deploying Web Applications

Remember that deployment is the last step in a process. The first is setting up the program — packaging the program into a component that is best suited for the deployment that follows. You can actually deploy a Web application in a number of ways. You can use the XCopy capability that simply wows audiences when demonstrated (because of its simplicity). A second method is to use Visual Studio 2008's capability to copy a Web site from one location to another using the Copy Web Site feature, as well as an alternative method that uses Visual Studio to deploy a precompiled Web application. The final method uses Visual Studio to build an installer program that can be launched on another machine. After reviewing each of the available methods, you can decide which is best for what you are trying to achieve. Start by looking at the simplest of the three methods: XCopy.

Using XCopy

Because of the nature of the .NET Framework, it is considerably easier to deploy .NET applications now than it was to deploy applications constructed using Microsoft's predecessor technology — COM. Applications in .NET compile down to assemblies, and these assemblies contain code that is executed by the Common Language Runtime (CLR). One great thing about assemblies is that they are self-describing. All the details about the assembly are stored within the assembly itself. In the Windows DNA world, COM stored all its self-describing data within the server's registry, so installing (as well as uninstalling) COM components meant shutting down IIS. Because a .NET assembly stores this information within itself, XCOPY deployment is possible and no registry settings are needed. Installing an assembly is as simple as copying it to another server and you do not need to stop or start IIS while this is going on.

We mention XCOPY here because it is the command-line way of basically doing a copy-and-paste of the files you want to move. XCOPY, however, provides a bit more functionality than just a copy-and-paste, as you will see shortly. XCOPY enables you to move files, directories, and even entire drives from one point to another.

The default syntax of the XCOPY command is as follows:

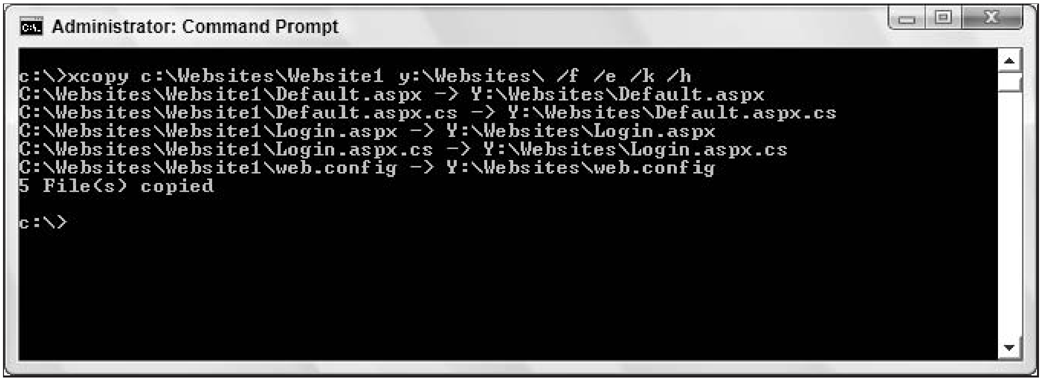
```
xcopy  [source]  [destination]  [/w]  [/p]  [/c]  [/v]  [/q]  [/f]  [/l]  [/g]
```

```
[/d[:mm-dd-yyyy]]  [/u]  [/i]  [/s  [/e]]  [/t]  [/k]  [/r]  [/h]  [{/a|/m}]  [/n]  [/o]
[/x] [/exclude:file1[+[file2]][+file3]] [{/y|/-y}] [/z]
```

To see an example of using the XCOPY feature, suppose you are working from your developer machine (`C:\`) and want to copy your ASP.NET application to a production server (`Z:\`). In its simplest form, the following command would do the job:

```
xcopy c:\Websites\Website1 y:\Websites\ /f /e /k /h
```

This move copies the files and folders from the source drive to the destination drive. Figure 34-2 shows an example of this use on the command line.



```
c:\>xcopy c:\Websites\Website1 y:\Websites\ /f /e /k /h
C:\Websites\Website1\Default.aspx -> Y:\Websites\Default.aspx
C:\Websites\Website1\Default.aspx.cs -> Y:\Websites\Default.aspx.cs
C:\Websites\Website1\Login.aspx -> Y:\Websites\Login.aspx
C:\Websites\Website1\Login.aspx.cs -> Y:\Websites\Login.aspx.cs
C:\Websites\Website1\web.config -> Y:\Websites\web.config
5 File(s) copied

c:\>
```

Figure 34-2

When you copy files using XCOPY, be aware that this method does not allow for the automatic creation of any virtual directories in IIS. When copying a new web application, you also need to create a virtual directory in the destination server and associate this virtual directory with the application you are copying. It is a simple process, but you must take these extra steps to finalize the site copy actions.

You can provide a number of parameters to this XCOPY command to get it to behave as you want it to. The following table details these parameters.

Parameter	Description
`/w`	Displays the message: `Press any key to begin copying file(s)`. It waits for your response to start the copying process.
`/p`	Asks for a confirmation on each file being copied. This is done in a file-by-file manner.
`/c`	Ignores errors that might occur in the copying process.
`/v`	Performs a verification on the files being copied to make sure they are identical to the source files.
`/q`	Suppresses any display of the XCOPY messages.
`/f`	Displays the file names for the source and destination files while the copying process is occurring.

Parameter	Description
`/l`	Displays a list of the files to be copied to the destination drive.
`/g`	Builds decrypted files for the destination drive.
`/d`	When used as simply `/d`, the only files copied are those newer than the existing files located in the destination location. Another alternative is to use `/d[:mm-dd-yyyy]`, which copies files that have been modified either on or after the specified date.
`/u`	Copies only source files that already exist in the destination location.
`/i`	If what is being copied is a directory or a file that contains wildcards and the same item does not exist in the destination location, a new directory is created. The XCOPY process also copies all the associated files into this directory.
`/s`	Copies all directories and their subdirectories only if they contain files. All empty directories or subdirectories are not copied in the process.
`/e`	Copies all subdirectories regardless of whether these directories contain files.
`/t`	Copies the subdirectories only and not the files they might contain.
`/k`	By default, the XCOPY process removes any read-only settings that might be contained in the source files. Using `/k` ensures that these read-only settings remain in place during the copying process.
`/r`	Copies only the read-only files to the destination location.
`/h`	Specifies that the hidden and system files, which are usually excluded by default, are included.
`/a`	Copies only files that have their archive file attributes set, and leaves the archive file attributes in place at the XCOPY destination.
`/m`	Copies only files that have their archive file attributes set, and turns off the archive file attributes.
`/n`	Copies using the NTFS short file and short directory names.
`/o`	Copies the discretionary access control list (DACL) in addition to the files.
`/x`	Copies the audit settings and the system access control list (SACL) in addition to the files.
`/exclude`	Allows you to exclude specific files. The construction used for this is `exclude: File1.aspx` + `File2.aspx` + `File3.aspx`.
`/y`	Suppresses any prompts from the XCOPY process that ask whether to overwrite the destination file.
`/-y`	Adds prompts in order to confirm an overwrite of any existing files in the destination location.
`/z`	Copies files and directories over a network in restartable mode.
`/?`	Displays help for the XCOPY command.

Using XCOPY is an easy way to move your applications from one server to another with little work on your part. If you have no problem setting up your own virtual directories, this mode of deployment should work just fine for you.

When the Web application is copied (and if placed in a proper virtual directory), it is ready to be called from a browser.

Using the VS Copy Web Site Option

The next option for copying a Web site is to use a GUI provided by Visual Studio 2008. This Copy Web Site GUI enables you to copy Web sites from your development server to either the same server or a remote server (as you can when you use the XCOPY command).

You can pull up this Copy Web Site dialog in Visual Studio in two ways. The first way is to click in the Copy Web Site icon in the Visual Studio Solution Explorer. This icon is shown in Figure 34-3.

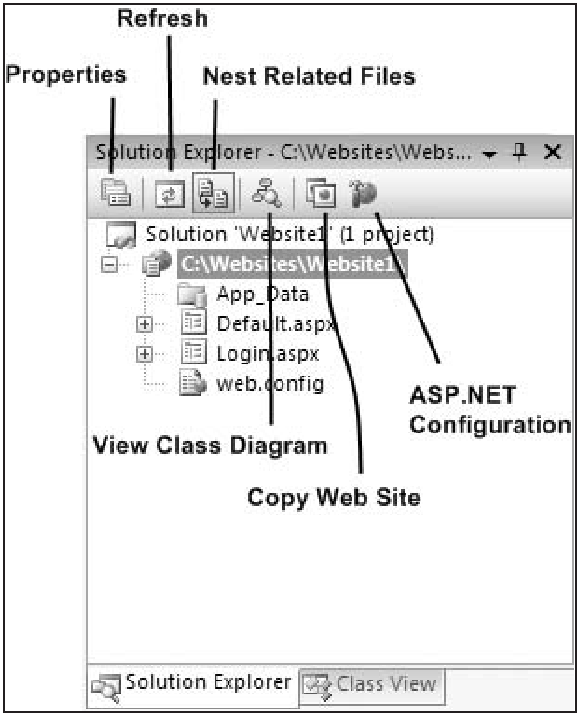
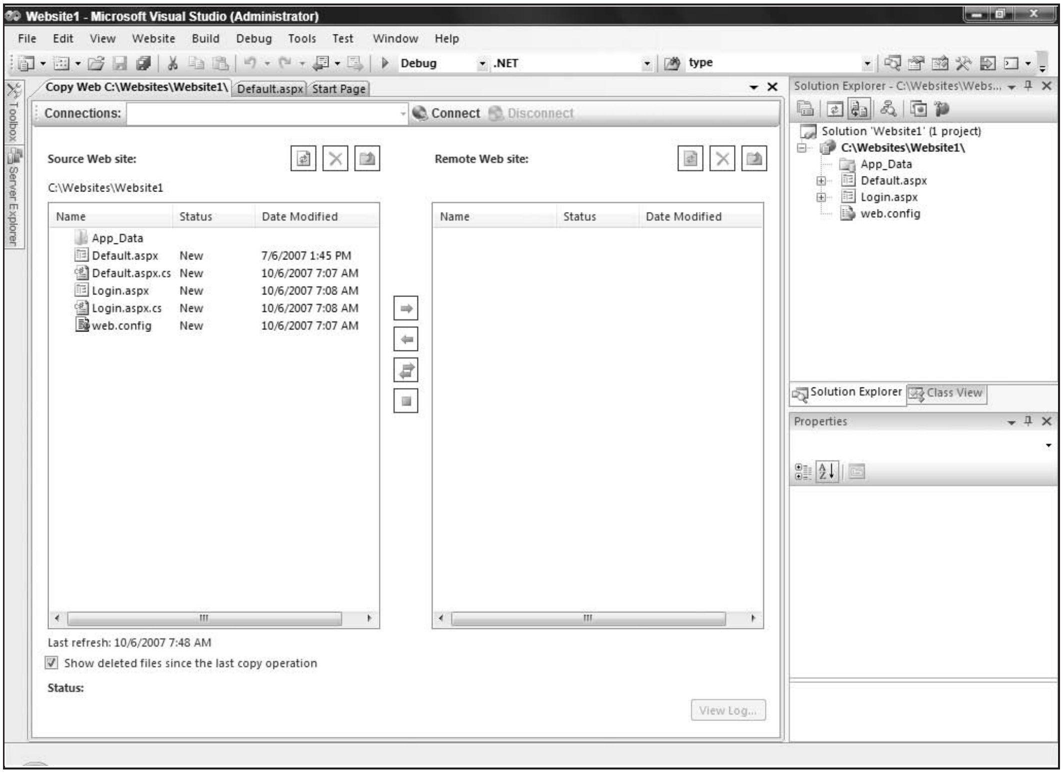

Refresh
Properties
Nest Related Files
Solution Explorer - C:\Websites\Webs...
Solution 'Website1' (1 project)
C:\Websites\Website1\
App_Data
Default.aspx
Login.aspx
web.config
ASP.NET Configuration
View Class Diagram
Copy Web Site
Solution Explorer
Class View

Figure 34-3

The other way to open the Copy Web Site GUI is to choose Website ➪ Copy Web Site from the Visual Studio menu. Using either method pulls up the Copy Web Site GUI in the Document window, as illustrated in Figure 34-4.

From this GUI, you can click the Connect To a Remote Server button (next to the Connections text box). This action brings up the Open Web Site dialog shown in Figure 34-5.


Website1 - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Tools Test Window Help
Debug
.NET
type
Toolbox
Server Explorer
Copy Web C:\Websites\Website1\
Default.aspx
Start Page
Connections:
Connect
Disconnect
Source Web site:
Remote Web site:
View Log...
Solution Explorer - C:\Websites\Webs...
Solution 'Website1' (1 project)
C:\Websites\Website1\
App_Data
Default.aspx
Login.aspx
web.config
Solution Explorer
Class View
Properties

C:\Websites\Website1

Name	Status	Date Modified
App_Data		
Default.aspx	New	7/6/2007 1:45 PM
Default.aspx.cs	New	10/6/2007 7:07 AM
Login.aspx	New	10/6/2007 7:08 AM
Login.aspx.cs	New	10/6/2007 7:08 AM
web.config	New	10/6/2007 7:07 AM

Last refresh: 10/6/2007 7:48 AM

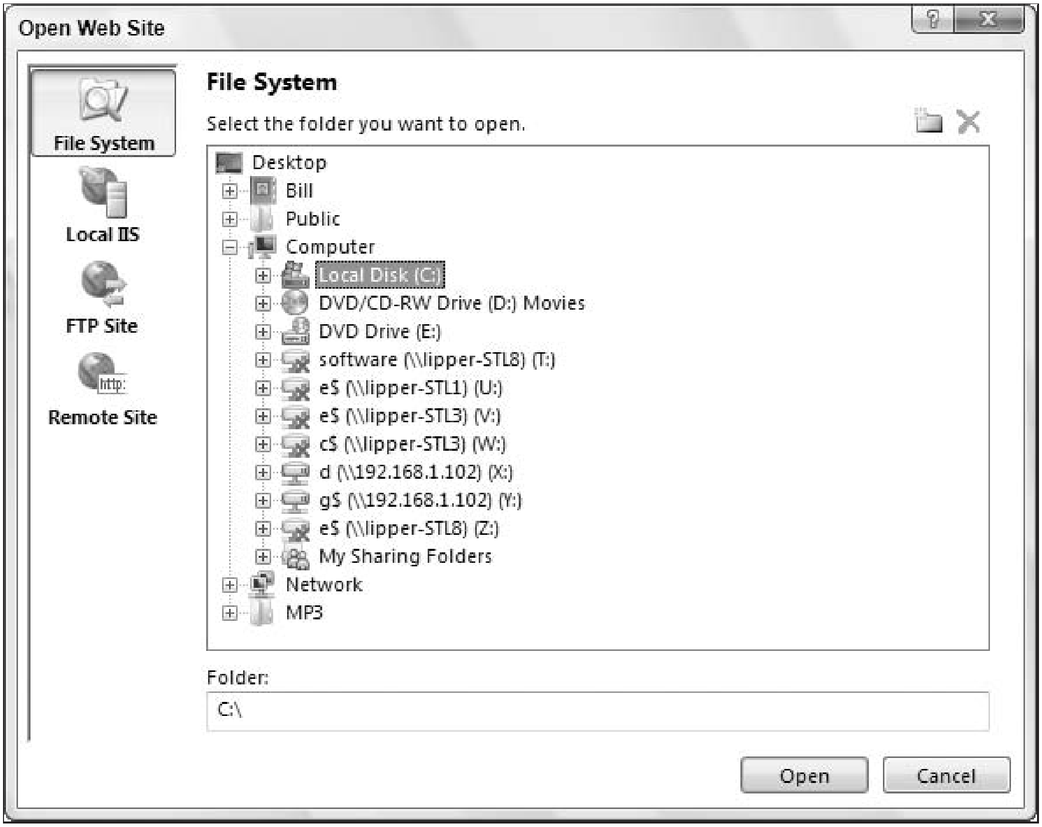
☑ Show deleted files since the last copy operation

Status:

Figure 34-4

As you can see from this dialog, you have a couple of options to connect to and copy your Web application. These options include the following:

- **File System:** This option allows you to navigate through a file explorer view of the computer. If you are going to install on a remote server from this view, you must have already mapped a drive to the installation location.
- **Local IIS:** This option enables you to use your local IIS in the installation of your Web application. From this part of the dialog, you can create new applications as well as new virtual directories directly. You can also delete applications and virtual directories from the same dialog. The Local IIS option does not permit you to work with IIS installations on any remote servers.
- **FTP Site:** This option enables you to connect to a remote server using FTP capabilities. From this dialog, you can specify the server that you want to contact using a URL or IP address, the port you are going to use, and the directory on the server that you will work with. From this dialog, you can also specify the username and password that may be required to access the server via FTP. Note that if you access this server with this dialog via FTP and provide a username and password, the items are transmitted in plain text.
- **Remote Site:** This option enables you to connect to a remote site using FrontPage Server Extensions. From this option in the dialog, you can also choose to connect to the remote server using Secure Sockets Layer (SSL).


Open Web Site
File System
Local IIS
FTP Site
Remote Site
File System
Select the folder you want to open.
Desktop
Bill
Public
Computer
Local Disk (C:)
DVD/CD-RW Drive (D:) Movies
DVD Drive (E:)
software (\\lipper-STL8) (T:)
e$ (\\lipper-STL1) (U:)
e$ (\\lipper-STL3) (V:)
c$ (\\lipper-STL3) (W:)
d (\\192.168.1.102) (X:)
g$ (\\192.168.1.102) (Y:)
e$ (\\lipper-STL8) (Z:)
My Sharing Folders
Network
MP3
Open
Cancel

Folder: C:\

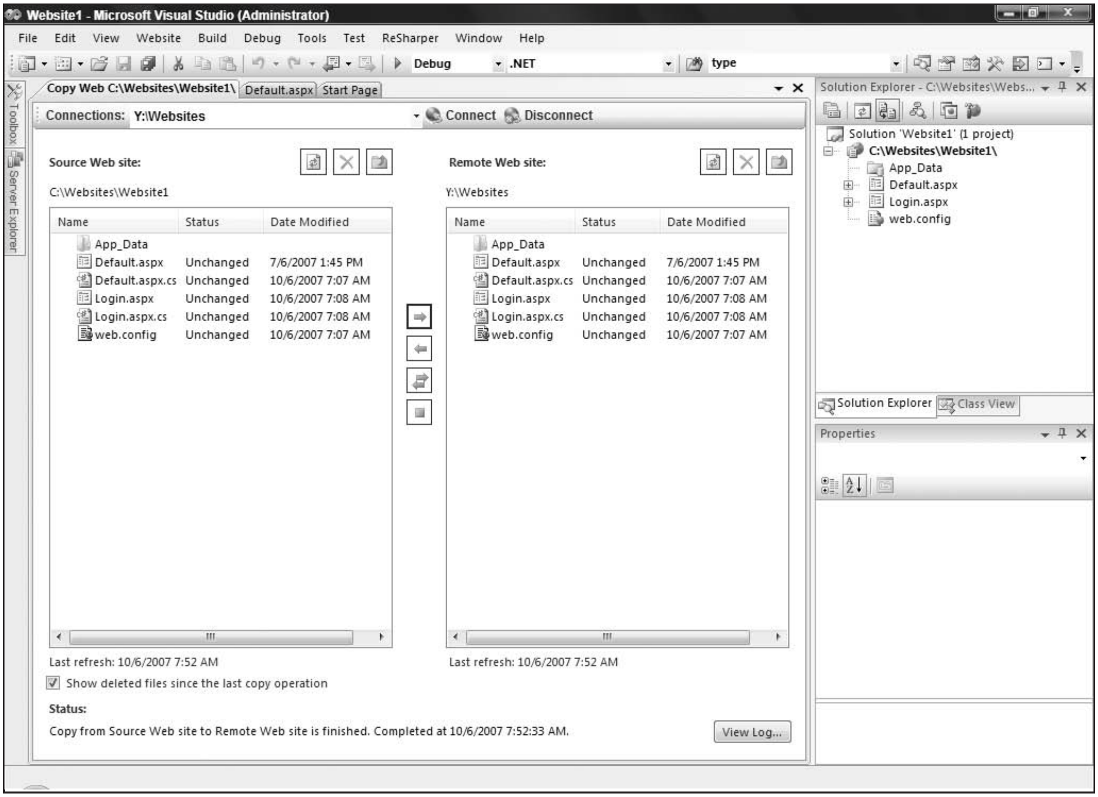
Figure 34-5

After being connected to a server, you can copy the contents of your Web application to it by selecting all or some of the files from the Source Web Site text area. After you select these files in the dialog, some of the movement arrows become enabled. Clicking the right-pointing arrow copies the selected files to the destination server. In Figure 34-6 you can see that, indeed, the files have been copied to the remote destination.

If you pull up the same copy dialog later, after working on the files, you see an arrow next to the files that have been changed in the interim and are, therefore, newer than those on the destination server (see Figure 34-7).

These arrows enable you to select only the files that must be copied again and nothing more. All the copying actions are recorded in a log file. You can view the contents of this log file from the Copy Web Site dialog by clicking the View Log button at the bottom of the dialog. This pulls up the `CopyWebSite.log` text file. From the copy that you made previously, you can see the transaction that was done. An example log entry is shown here:

```
Copy from 'C:\Websites\Website1' to 'Y:\Websites' started at 10/6/2007 7:52:31 AM.
        Create folder App_Data in the remote Web site.
        Copy file Default.aspx from source to remote Web site.
        Copy file Default.aspx.cs from source to remote Web site.
        Copy file Login.aspx from source to remote Web site.
        Copy file Login.aspx.cs from source to remote Web site.
        Copy file web.config from source to remote Web site.

Copy from 'C:\Websites\Website1' to 'Y:\Websites' is finished. Completed at 10/6/2007
7:52:33 AM.
```


Website1 - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Tools Test ReSharper Window Help
Debug
.NET
type
Copy Web C:\Websites\Website1\
Default.aspx
Start Page
Toolbox
Server Explorer
Connect
Disconnect
View Log...
Solution Explorer - C:\Websites\Webs...
Solution 'Website1' (1 project)
C:\Websites\Website1\
App_Data
Default.aspx
Login.aspx
web.config
Solution Explorer
Class View
Properties

Connections: Y:\Websites

Source Web site:

C:\Websites\Website1

Name	Status	Date Modified
App_Data		
Default.aspx	Unchanged	7/6/2007 1:45 PM
Default.aspx.cs	Unchanged	10/6/2007 7:07 AM
Login.aspx	Unchanged	10/6/2007 7:08 AM
Login.aspx.cs	Unchanged	10/6/2007 7:08 AM
web.config	Unchanged	10/6/2007 7:07 AM

Last refresh: 10/6/2007 7:52 AM

Remote Web site:

Y:\Websites

Name	Status	Date Modified
App_Data		
Default.aspx	Unchanged	7/6/2007 1:45 PM
Default.aspx.cs	Unchanged	10/6/2007 7:07 AM
Login.aspx	Unchanged	10/6/2007 7:08 AM
Login.aspx.cs	Unchanged	10/6/2007 7:08 AM
web.config	Unchanged	10/6/2007 7:07 AM

Last refresh: 10/6/2007 7:52 AM

☑ Show deleted files since the last copy operation

Status:

Copy from Source Web site to Remote Web site is finished. Completed at 10/6/2007 7:52:33 AM.

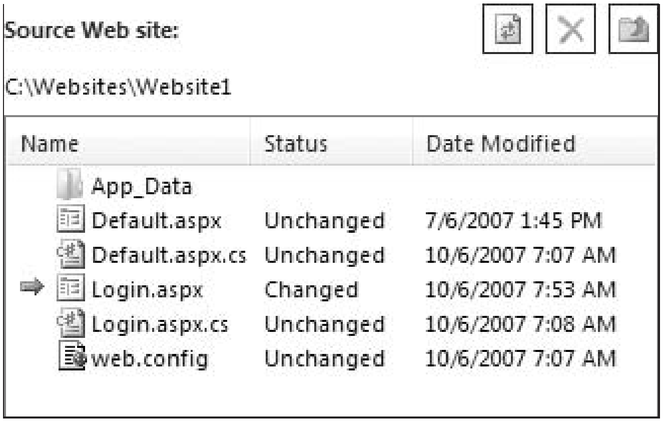
Figure 34-6

Deploying a Precompiled Web Application

In addition to using Visual Studio to copy a Web application from one location to another, it is also possible to use this IDE to deploy a precompiled application. The process of precompiling a Web application is explained in Chapter 1. ASP.NET 3.5 includes a precompilation process that allows for a process referred to as *precompilation for deployment*.

What happens in the precompilation for deployment process is that each page in the Web application is built and compiled into a single application DLL and some placeholder files. These files can then be deployed together to another server and run from there. The nice thing about this precompilation process is that it obfuscates your code by placing all page code (as well as the page's code-behind code) into the

DLL, thereby making it more difficult for your code to be stolen or changed if you select this option in the compilation process. This is an ideal situation when you are deploying applications your customers are paying for, or applications that you absolutely do not want changed in any manner after deployment.



Source Web site:

C:\Websites\Website1

Name	Status	Date Modified
App_Data		
Default.aspx	Unchanged	7/6/2007 1:45 PM
Default.aspx.cs	Unchanged	10/6/2007 7:07 AM
Login.aspx	Changed	10/6/2007 7:53 AM
Login.aspx.cs	Unchanged	10/6/2007 7:08 AM
web.config	Unchanged	10/6/2007 7:07 AM

Figure 34-7

Chapter 1 showed you how to use the command-line tool `aspnet_compiler.exe` to accomplish the task of precompilation. Although this is a great method for precompiling your Web applications and deploying them to remote servers, you can also use Visual Studio 2008 to accomplish the precompilation and deployment process.

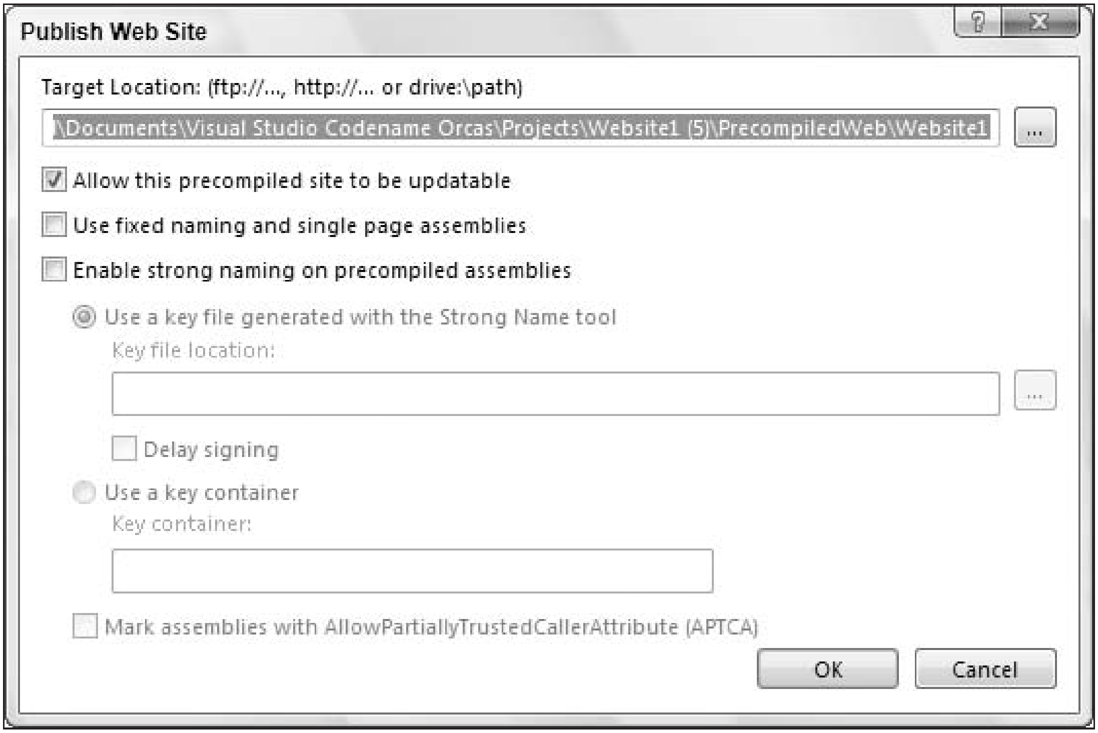
To accomplish this task, open up the project you want to deploy and get the application ready for deployment by turning off the debugging capabilities as described earlier in the chapter. Then pull up the precompilation and deployment dialog by choosing Build ➪ Publish Web Site in the Visual Studio menu. This opens the Publish Web Site dialog shown in Figure 34-8.

Using the Browse (. . .) button in this dialog, you can choose any remote location to which you want to deploy the application. As in earlier examples, your options are a file system location, a place in the local IIS, a location accessed using FTP, or a location accessed via FrontPage Server Extensions.

Other options in this dialog include the Allow this precompiled site to be updateable check box. When checked, the site will be compiled and copied without any changes to the `.aspx` pages. This means that after the precompilation process, you can still make minor changes to the underlying pages and the application will work and function as normal. If this check box is unchecked, all the code from the pages is stripped out and placed inside a single DLL. In this state, the application is not updateable because it is impossible to update any of the placeholder files from this compilation process.

Another option in this dialog is to assign a strong name to the DLL that is created in this process. You can select the appropriate check box and assign a key to use in the signing process. The created DLL from the precompilation will then be a strong-assembly — signed with the key of your choice.

When you are ready to deploy, click OK in the dialog and then the open application is built and published. *Published* means that the application is deployed to the specified location. Looking at this location, you can see that a `bin` directory has now been added that contains the precompiled DLL, which is your Web application. This is illustrated in Figure 34-9.



Target Location: (ftp://..., http://... or drive:\path)

\Documents\Visual Studio Codename Orcas\Projects\Website1 (5)\PrecompiledWeb\Website1

☑ Allow this precompiled site to be updatable

☐ Use fixed naming and single page assemblies

☐ Enable strong naming on precompiled assemblies

● Use a key file generated with the Strong Name tool

Key file location:

☐ Delay signing

○ Use a key container

Key container:

☐ Mark assemblies with AllowPartiallyTrustedCallerAttribute (APTCA)

Figure 34-8

In this state, the code contained in any of the ASP.NET-specific pages is stripped out and placed inside the DLL. The files that you see are actually just placeholders that the DLL needs for reference.

Building an Installer Program

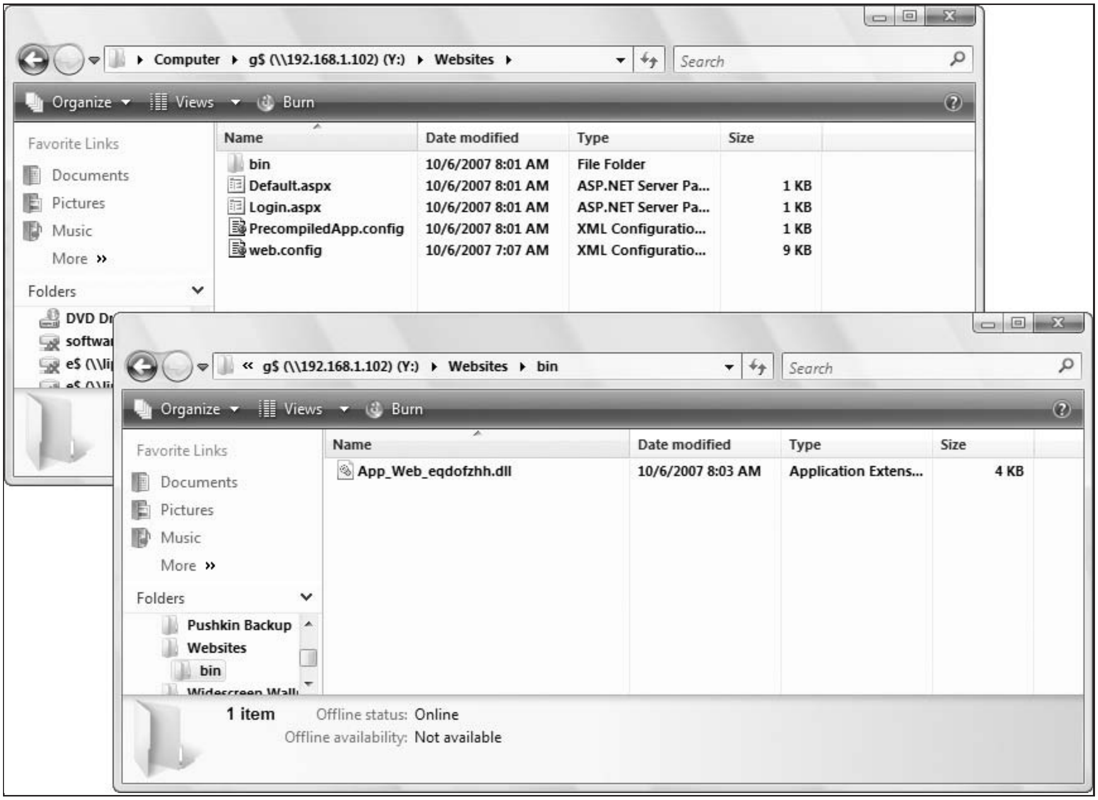
The final option you should look at is how to use Visual Studio to build an installation program. After the program is constructed, a consumer can run the installation program on a server where it performs a series of steps to install the Web application.

Packaging your Web application into an installer program works in many situations. For instance, if you sell your Web application, one of the simpler ways for the end user to receive the application is as an executable that can be run on the computer and installed — all without much effort on his part.

The Windows Installer

The Windows Installer service was introduced with Windows 2000, although it is also available in Windows XP, Windows Server 2003, and Windows Server 2008. This service was introduced to make the installation process for your Windows-based applications as easy as possible.

You use the Windows Installer technology not only for ASP.NET applications but also for any type of Windows-based application. The Windows Installer service works by creating a set of rules that determine how the application is to be installed. These rules are packaged into a Windows Installer Package File that uses the `.msi` file extension.



Computer ▸ g$ (\\192.168.1.102) (Y:) ▸ Websites ▸
Search
Organize
Views
Burn
Favorite Links
Documents
Pictures
Music
More »
Folders
DVD Dr
softwa
e$ (\\li

Name	Date modified	Type	Size
bin	10/6/2007 8:01 AM	File Folder	
Default.aspx	10/6/2007 8:01 AM	ASP.NET Server Pa...	1 KB
Login.aspx	10/6/2007 8:01 AM	ASP.NET Server Pa...	1 KB
PrecompiledApp.config	10/6/2007 8:01 AM	XML Configuratio...	1 KB
web.config	10/6/2007 7:07 AM	XML Configuratio...	9 KB

« g$ (\\192.168.1.102) (Y:) ▸ Websites ▸ bin
Search
Organize
Views
Burn
Favorite Links
Documents
Pictures
Music
More »
Folders
Pushkin Backup
Websites
bin

Name	Date modified	Type	Size
App_Web_eqdofzhh.dll	10/6/2007 8:03 AM	Application Extens...	4 KB

1 item Offline status: Online
Offline availability: Not available

Figure 34-9

The Windows Installer service considers all applications to be made up of three parts:

- **Products:** The large-bucket item being installed, also known as the application itself. An example of this is the ASP.NET Web application.
- **Features:** Features are subsets of products. Products are made up of one or more features.
- **Components:** Components make up features. A feature is made up of one or more components. A single component can be utilized by several features in the product.

The Windows Installer service is a powerful offering and can be modified in many ways. Not only does the Windows Installer technology detail the product, features, and components of what is to be installed, but it can also take other programmatic actions or show a sequence of user interfaces as the installation process proceeds. For detailed information on the Windows Installer, be sure to view the MSDN documentation on the Windows Installer SDK.

With that said, working with the Windows Installer SDK is complicated at best; that was the reason for the release of the Visual Studio Installer (VSI) as an add-on with Visual Studio 6. This addition made the steps for building an installer much easier to follow. Visual Studio 2008 continues to expand on this capability. You have quite a few options for the deployment projects you can build with Visual Studio 2008. Such projects include the following:

- **Setup Project:** This project type allows you to create a standard Windows Installer setup for a Windows application.
- **Web Setup Project:** This is the project type covered in this chapter. It's the type of setup project you use to create an installer for an ASP.NET Web application.
- **Merge Module Project:** This project type creates a merge module similar to a cabinet file. A merge module, such as a cabinet file, allows you to package a group of files for distribution but not for installation. The idea is that you use a merge module file with other setup programs. This project type produces a file type with an extension of `.msm`.
- **Setup Wizard:** This selection actually gives you a wizard to assist you through one of the other defined project types.
- **Cab Project:** This project type creates a cabinet file (`.cab`) that packages a group of files for distribution. It is similar to a merge module file, but the cabinet file is different in that it allows for installation of the files contained in the package.
- **Smart Device Cab Project:** This new project type allows for the creation of a cabinet file that is installed on a smart device instead of on a typical operating system.

Although you have a number of different setup and deployment project types at your disposal, the Web Setup Project is the only one covered in this chapter because it is the project you use to build an installer for an ASP.NET Web application.

Actions of the Windows Installer

You might already be thinking that using the Windows Installer architecture for your installation program seems a lot more complicated than using the methods shown previously in this chapter. Yes, it is a bit more complicated — mainly because of the number of steps required to get the desired result; but in the end, you are getting a lot more control over how your applications are installed.

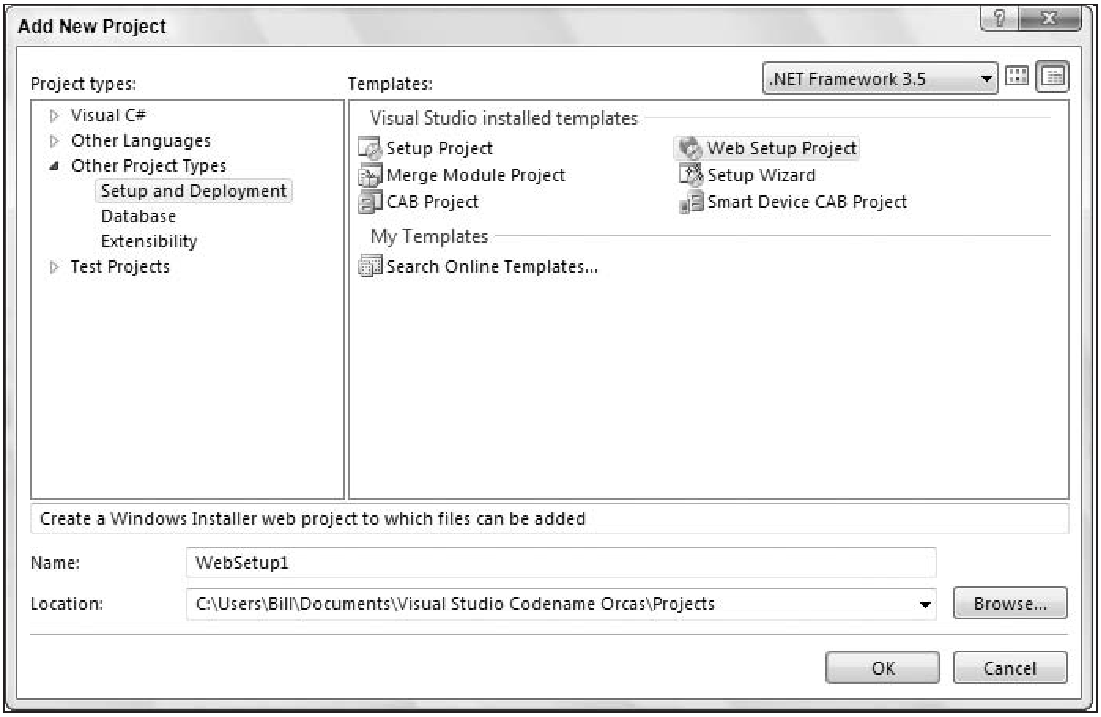
Using an installer program gives you programmatic logic over how your applications are installed. You also gain other advantages, such as:

- The capability to check if the .NET Framework is installed, as well as which version of the Framework is installed
- The capability to read or write values to the registry
- The capability to collect information from the end user during the installation process
- The capability to run scripts
- The capability to include such features such as dialogs and splash screens during the installation process

Creating a Basic Installation Program

You can apply a tremendous amount of customization to the installation programs you build. Let's start, however, by looking at how to create a basic installation program for your ASP.NET Web application. To create an installer for your application, first open up the project for which you want to create a deployment project in Visual Studio. The next step is to add an installer program to the solution. To do this, you add the setup program as a new project contained within the same solution. Choose File ➪ New ➪ Project from the Visual Studio menu. This launches the New Project dialog.

From the New Project dialog, first expand Other Project Types from the left-hand pane in the dialog and then select Setup and Deployment. This provides you with a list of all the available setup and deployment projects in Visual Studio. For our purposes, select Web Setup Project (shown in Figure 34-10).


Add New Project
Project types:
Visual C#
Other Languages
Other Project Types
Setup and Deployment
Database
Extensibility
Test Projects
Templates:
.NET Framework 3.5
Visual Studio installed templates
Setup Project
Merge Module Project
CAB Project
Web Setup Project
Setup Wizard
Smart Device CAB Project
My Templates
Search Online Templates...
Browse...
OK
Cancel

Create a Windows Installer web project to which files can be added

Name: WebSetup1

Location: C:\Users\Bill\Documents\Visual Studio Codename Orcas\Projects

Figure 34-10

Clicking OK in this dialog adds the Web Setup Project type to your solution. It uses the default name of `WebSetup1`. Visual Studio also opens up the File System Editor in the document window, which is shown in Figure 34-11.

The File System Editor shows a single folder: the Web Application Folder. This is a representation of what is going to be installed on the target machine. The first step is to add the files from the WebSite1 project to this folder. You do this by choosing Project ➪ Add ➪ Project Output from the Visual Studio menu. This pulls up the Add Project Output Group dialog. This dialog (shown in Figure 34-12) enables you to select the items you want to include in the installer program.

From this dialog, you can see that the project, Wrox, is already selected. Highlight the Content Files option and click OK. This adds all the files from the Wrox project to the WebSetup1 installer program. This addition is then represented in the File System Editor as well.

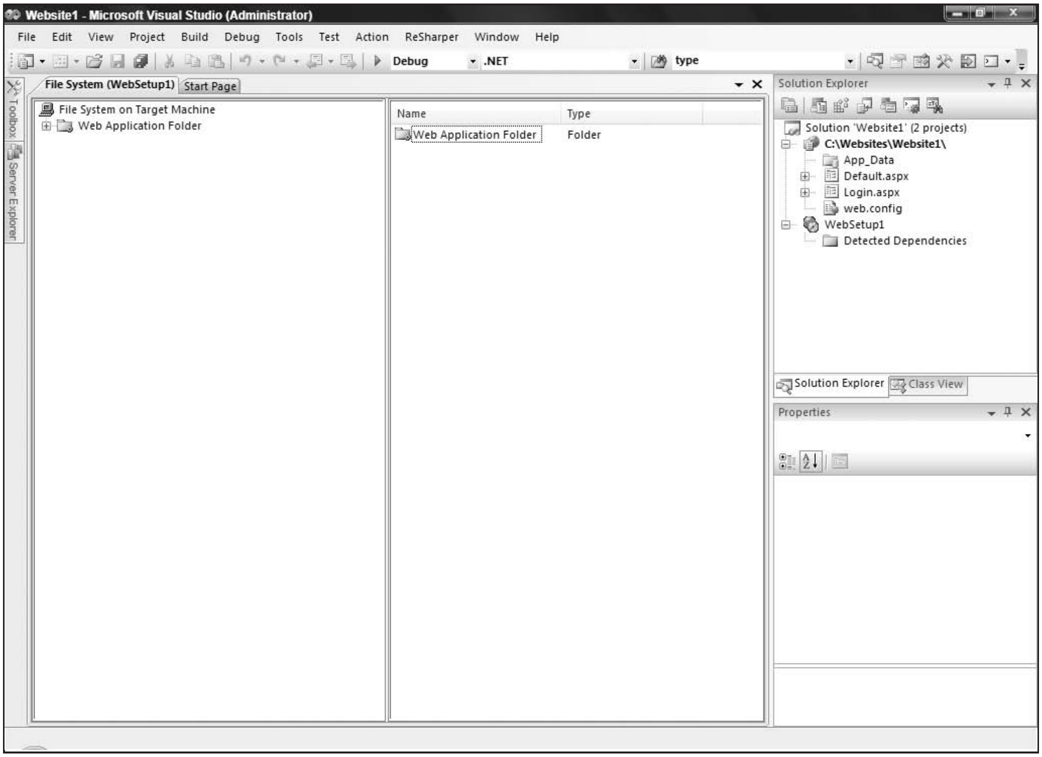

Website1 - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Tools Test Action ReSharper Window Help
Debug
.NET
type
File System (WebSetup1)
Start Page
Toolbox
Server Explorer
File System on Target Machine
Web Application Folder
Name
Type
Web Application Folder
Folder
Solution Explorer
Solution 'Website1' (2 projects)
C:\Websites\Website1\
App_Data
Default.aspx
Login.aspx
web.config
WebSetup1
Detected Dependencies
Solution Explorer
Class View
Properties

Figure 34-11

After the files are added to the installer program, the next step is to click the Launch Conditions Editor button in the Solution Explorer (see Figure 34-13) to open the editor. The Launch Conditions Editor is also displayed in Visual Studio's document window. From this editor, you can see that a couple of conditions are already defined for you. Obviously, for Web applications, it is important that IIS be installed. Logically, one of the defined conditions is that the program must perform a search to see if IIS is installed before installing the application. You should also stipulate that the installation server must have version 3.5 of the .NET Framework installed.

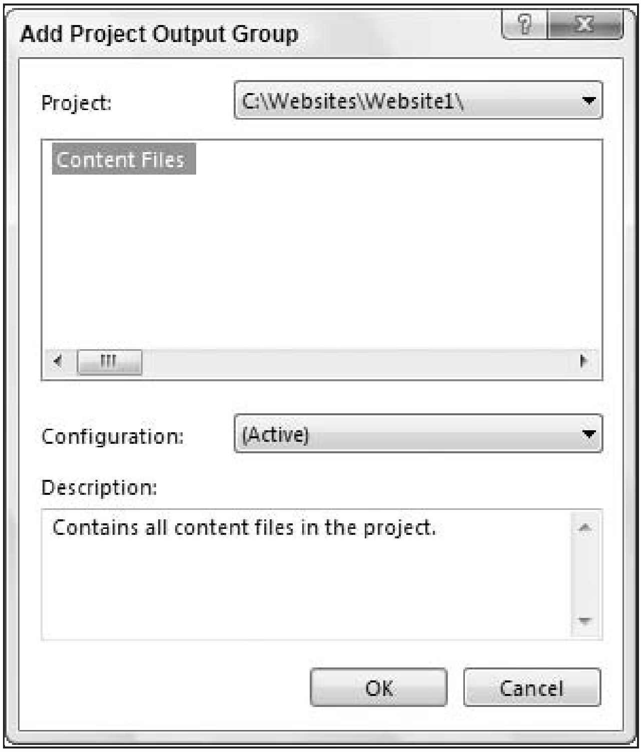
To establish this condition, right-click the Requirements On Target Machine node. Then select Add .NET Framework Launch Condition (as shown in Figure 34-14).

This adds the .NET Framework requirement to the list of launch conditions required for a successful installation of the Web application.

As a final step, highlight the WebSetup1 program in the Visual Studio Solution Explorer so you can modify some of the properties that appear in the Properties window. For now, you just change some of the self-explanatory properties, but you will review these again later in this chapter. For this example, however, just change the following properties:

- `Author`: Wrox
- `Description`: This is a test project.

- ❑ `Manufacturer`: Wrox
- ❑ `ManufacturerUrl`: `http://www.wrox.com`
- ❑ `SupportPhone`: 1-800-555-5555
- ❑ `SupportUrl`: `http://www.wrox.com/support/`



Project: C:\Websites\Website1\

Content Files

Configuration: (Active)

Description:

Contains all content files in the project.

Figure 34-12

Now the installation program is down to its simplest workable instance. Make sure Release is selected as the active solution configuration in the Visual Studio toolbar; then build the installer program by choosing Build ➪ Build WebSetup1 from the menu.

Looking in `C:\Documents and Settings\<username>\My Documents\Visual Studio 2008\Projects\Wrox\WebSetup1\Release`, you find the following files:

- ❑ `Setup.exe`: This is the installation program. It is meant for machines that do not have the Windows Installer service installed.
- ❑ `WebSetup1.msi`: This is the installation program for those that have the Windows Installer service installed on their machine.

That's it! You now have your ASP.NET Web application wrapped up in an installation program that can be distributed in any manner you want. It can then be run and installed automatically for the end user. Take a quick look in the following section at what happens when the consumer actually fires it up.

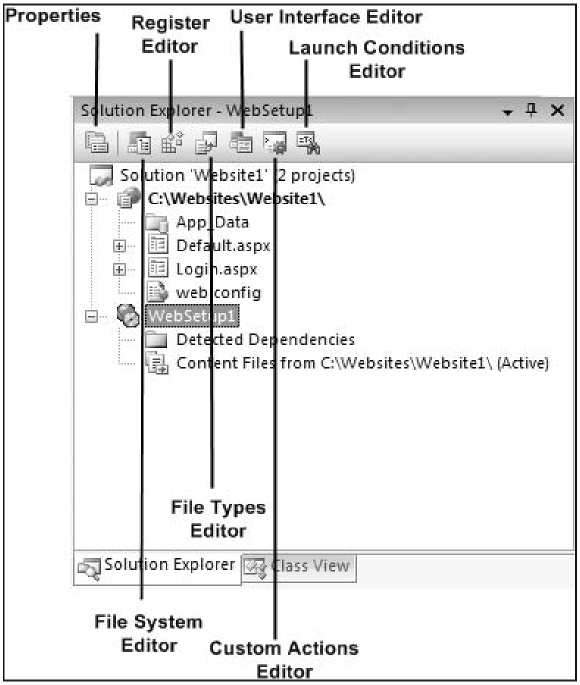

Properties
Register Editor
User Interface Editor
Launch Conditions Editor
Solution Explorer - WebSetup1
Solution 'Website1' (2 projects)
C:\Websites\Website1\
App_Data
Default.aspx
Login.aspx
web.config
WebSetup1
Detected Dependencies
Content Files from C:\Websites\Website1\ (Active)
File Types Editor
Solution Explorer
Class View
File System Editor
Custom Actions Editor

Figure 34-13

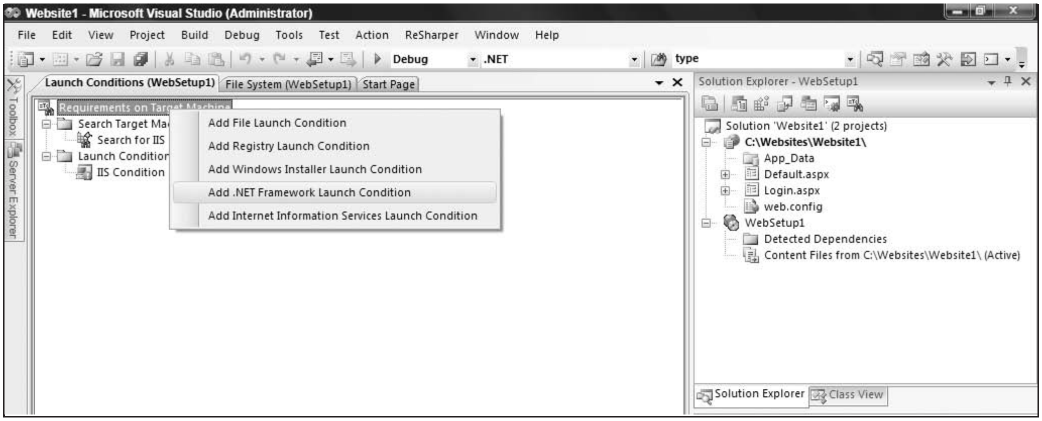
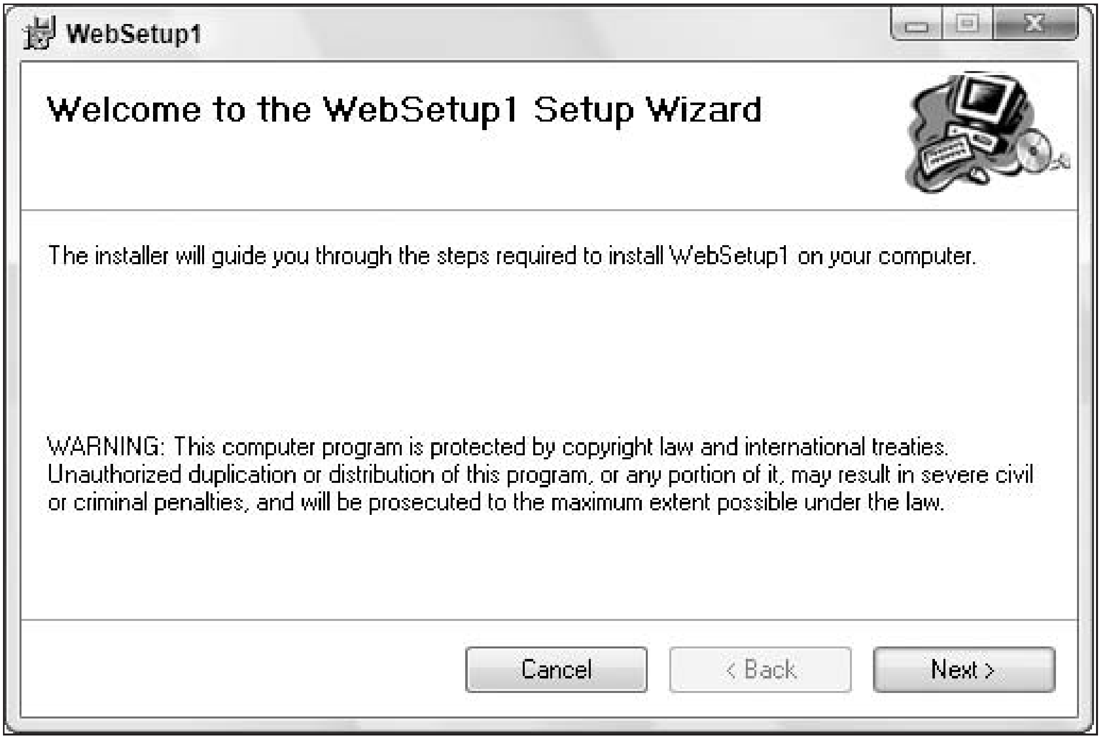

Website1 - Microsoft Visual Studio (Administrator)
File
Edit
View
Project
Build
Debug
Tools
Test
Action
ReSharper
Window
Help
Debug
.NET
type
Launch Conditions (WebSetup1)
File System (WebSetup1)
Start Page
Requirements on Target Machine
Search Target Ma
Search for IIS
Launch Conditior
IIS Condition
Add File Launch Condition
Add Registry Launch Condition
Add Windows Installer Launch Condition
Add .NET Framework Launch Condition
Add Internet Information Services Launch Condition
Toolbox
Server Explorer
Solution Explorer - WebSetup1
Solution 'Website1' (2 projects)
C:\Websites\Website1\
App_Data
Default.aspx
Login.aspx
web.config
WebSetup1
Detected Dependencies
Content Files from C:\Websites\Website1\ (Active)
Solution Explorer
Class View

Figure 34-14

Installing the Application

Installing the application is a simple process (as it should be). Double-click the `WebSetup1.msi` file to launch the installation program. This pulls up the Welcome screen shown in Figure 34-15.

From this dialog, you can see that the name of the program being installed is `WebSetup1`. Clicking Next gives you the screen shown in Figure 34-16.



Welcome to the WebSetup1 Setup Wizard

The installer will guide you through the steps required to install WebSetup1 on your computer.

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

Figure 34-15

This screen tells you what you are installing (the Default Web Site) as well as the name of the virtual directory created for the deployed Web application. The consumer can feel free to change the name of the virtual directory in the provided text box. A button in this dialog allows for an estimation of the disk cost (space required) for the installed application. In .NET 3.5, the installer also allows the end user to choose the application pool he or she is interested in using for the application. The next series of screens install the `WebSetup1` application (shown in Figure 34-17).

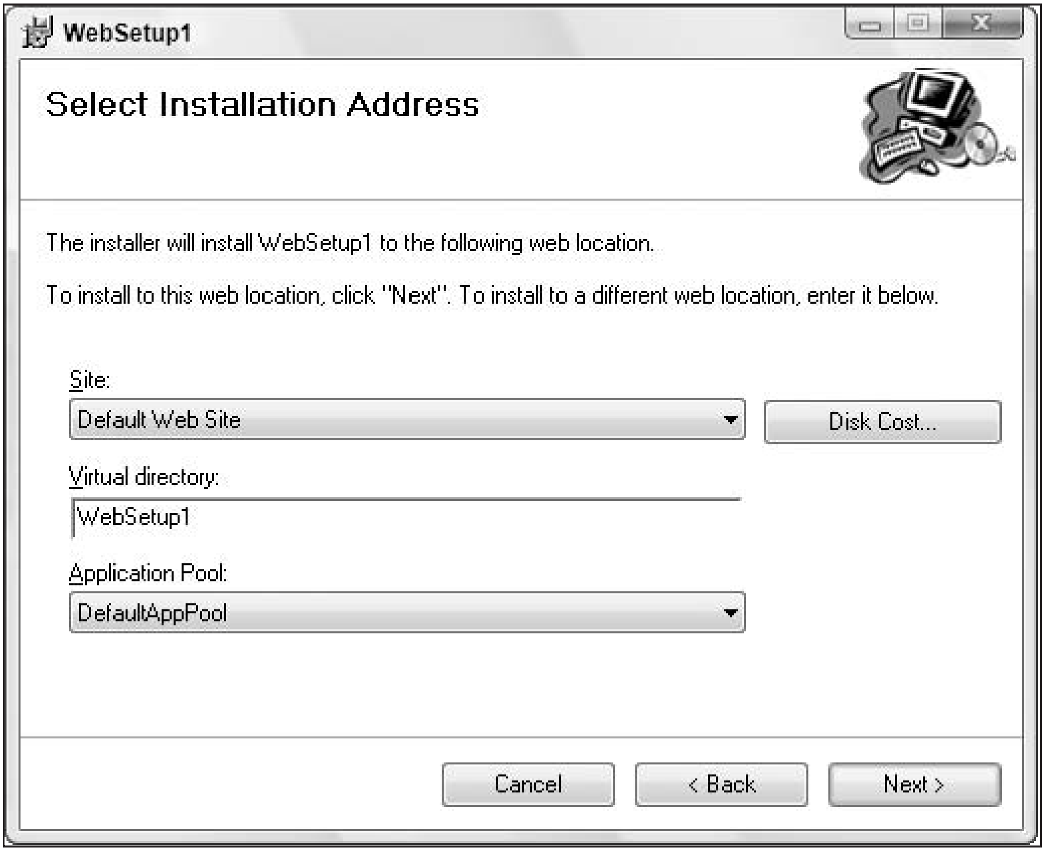
After the application is installed, you can find the `WebSetup1` folder and application files located in the `C:\Inetpub\wwwroot` folder (within IIS). The application can now be run on the server from this location.

Uninstalling the Application

To uninstall the application, the consumer has a couple of options. First, he can relaunch the `.msi` file and use the option to either repair the current installation or to remove the installation altogether (as shown in Figure 34-18).

The other option is to pull up the Add/Remove Programs dialog from the server's Control Panel. On the Control Panel, you see `WebSetup1` listed (as shown in Figure 34-19).

This dialog holds information about the size of the installed application and, if you are using Windows XP, it will also show you how often the application is used. Also, if you are using Windows XP, clicking



The installer will install WebSetup1 to the following web location.

To install to this web location, click "Next". To install to a different web location, enter it below.

Site:
Default Web Site

Virtual directory:
WebSetup1

Application Pool:
DefaultAppPool

Figure 34-16

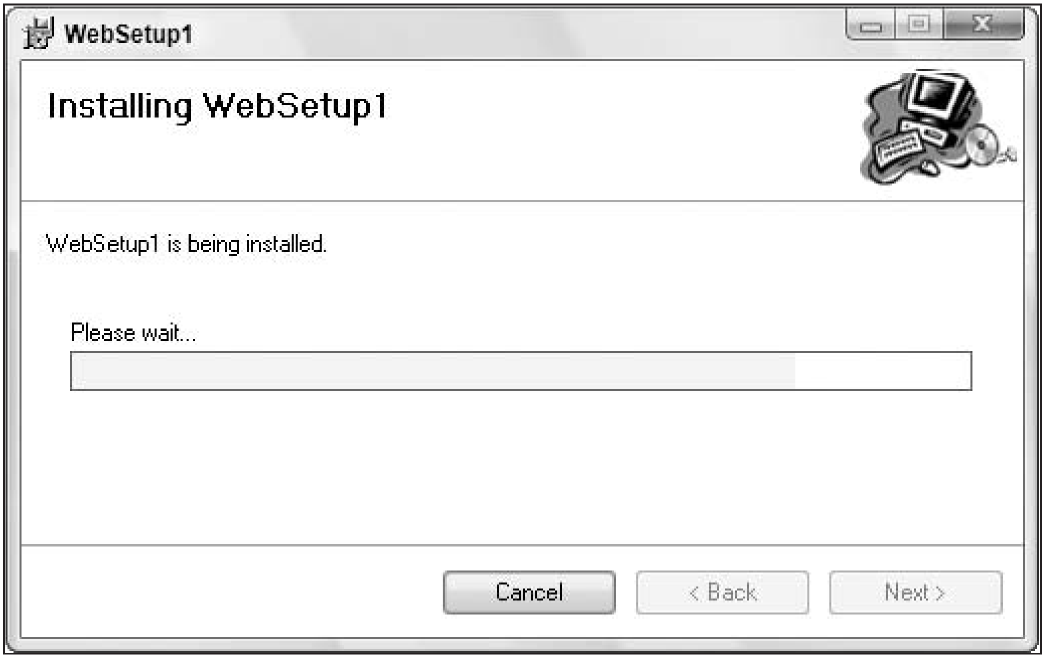
the support link pulls up the Support Info dialog, which shows the project's properties that you entered a little earlier (see Figure 34-20).

However, if you are using Windows Vista, you can get at the same information by right-clicking on the column headers and selecting the More option from the provided menu. This gives you a list of options (shown here in Figure 34-21), providing the same information as what you can see in Windows XP.

From the Add/Remove Programs dialog, you can remove the installation by clicking the Remove button of the selected program.

Looking More Closely at Installer Options

The Windows Installer service easily installs a simple ASP.NET Web application. The installer takes care of packaging the files into a nice `.msi` file from which it can then be distributed. Next, the `.msi` file takes care of creating a virtual directory and installing the application files. The installer also makes it just as easy to uninstall the application from the server. All these great services are provided with very little work on the user's part.

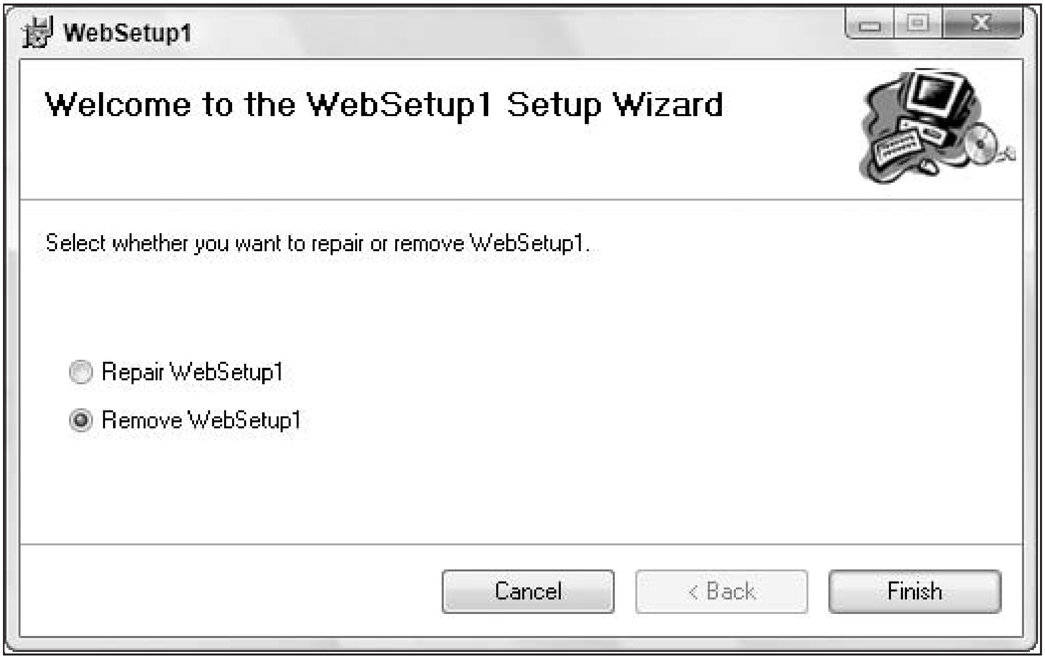

WebSetup1
Cancel
< Back
Next >

Installing WebSetup1

WebSetup1 is being installed.

Please wait...

Figure 34-17


WebSetup1
Cancel
< Back
Finish

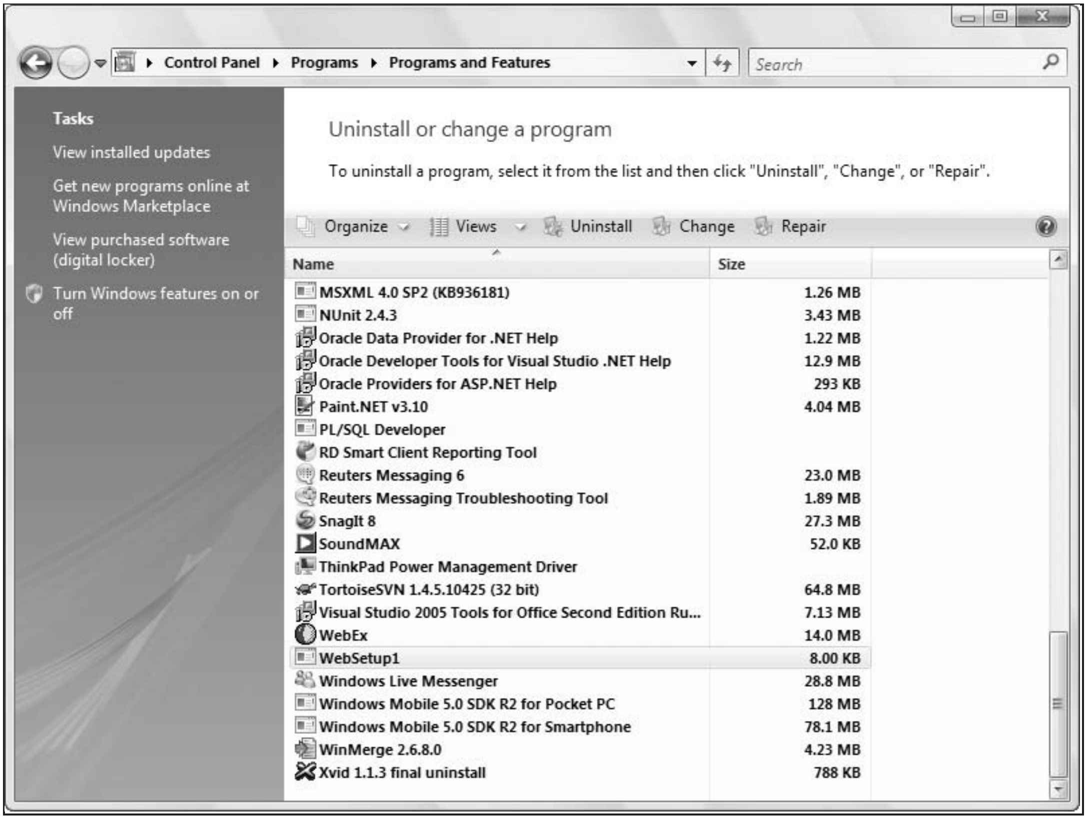
Welcome to the WebSetup1 Setup Wizard

Select whether you want to repair or remove WebSetup1.

○ Repair WebSetup1

● Remove WebSetup1

Figure 34-18

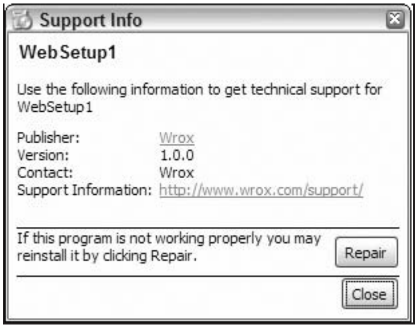

Control Panel ▸ Programs ▸ Programs and Features
Search
Tasks
View installed updates
Get new programs online at Windows Marketplace
View purchased software (digital locker)
Turn Windows features on or off
Organize
Views
Uninstall
Change
Repair

Uninstall or change a program

To uninstall a program, select it from the list and then click "Uninstall", "Change", or "Repair".

Name	Size
MSXML 4.0 SP2 (KB936181)	1.26 MB
NUnit 2.4.3	3.43 MB
Oracle Data Provider for .NET Help	1.22 MB
Oracle Developer Tools for Visual Studio .NET Help	12.9 MB
Oracle Providers for ASP.NET Help	293 KB
Paint.NET v3.10	4.04 MB
PL/SQL Developer	
RD Smart Client Reporting Tool	
Reuters Messaging 6	23.0 MB
Reuters Messaging Troubleshooting Tool	1.89 MB
SnagIt 8	27.3 MB
SoundMAX	52.0 KB
ThinkPad Power Management Driver	
TortoiseSVN 1.4.5.10425 (32 bit)	64.8 MB
Visual Studio 2005 Tools for Office Second Edition Ru...	7.13 MB
WebEx	14.0 MB
WebSetup1	8.00 KB
Windows Live Messenger	28.8 MB
Windows Mobile 5.0 SDK R2 for Pocket PC	128 MB
Windows Mobile 5.0 SDK R2 for Smartphone	78.1 MB
WinMerge 2.6.8.0	4.23 MB
Xvid 1.1.3 final uninstall	788 KB

Figure 34-19


Support Info
Repair
Close

WebSetup1

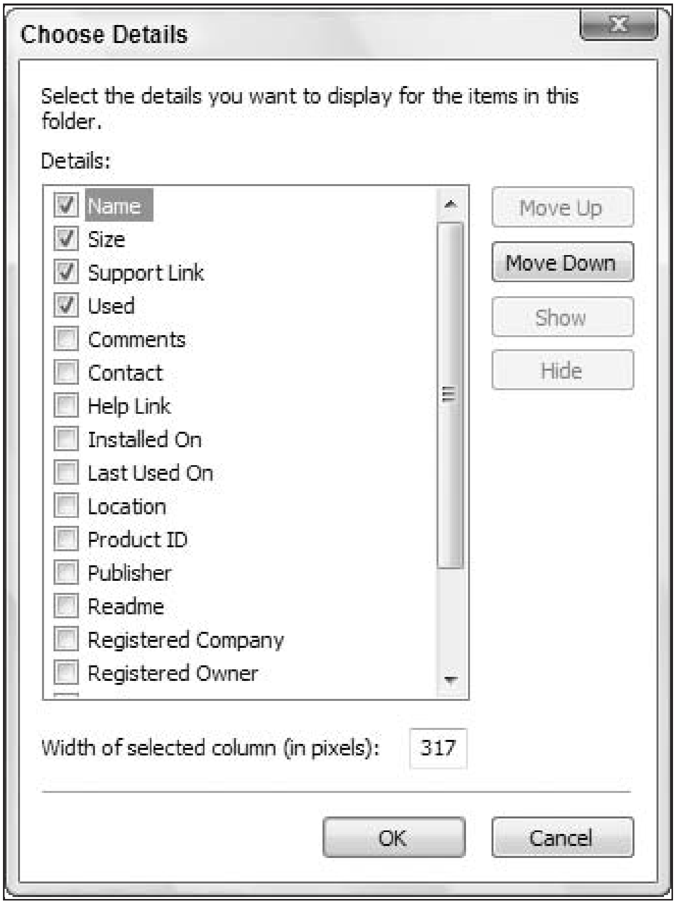
Use the following information to get technical support for WebSetup1

Publisher: Wrox
Version: 1.0.0
Contact: Wrox
Support Information: http://www.wrox.com/support/

If this program is not working properly you may reinstall it by clicking Repair.

Figure 34-20

Even though this approach addresses almost everything needed for an ASP.NET installer program, the setup and deployment project for Web applications provided by Visual Studio really provides much more in the way of options and customizations. This next section looks at the various ways you can work with modifying the installer program.



Select the details you want to display for the items in this folder.

Details:

- ☑ Name
- ☑ Size
- ☑ Support Link
- ☑ Used
- ☐ Comments
- ☐ Contact
- ☐ Help Link
- ☐ Installed On
- ☐ Last Used On
- ☐ Location
- ☐ Product ID
- ☐ Publisher
- ☐ Readme
- ☐ Registered Company
- ☐ Registered Owner

Width of selected column (in pixels): 317

Figure 34-21

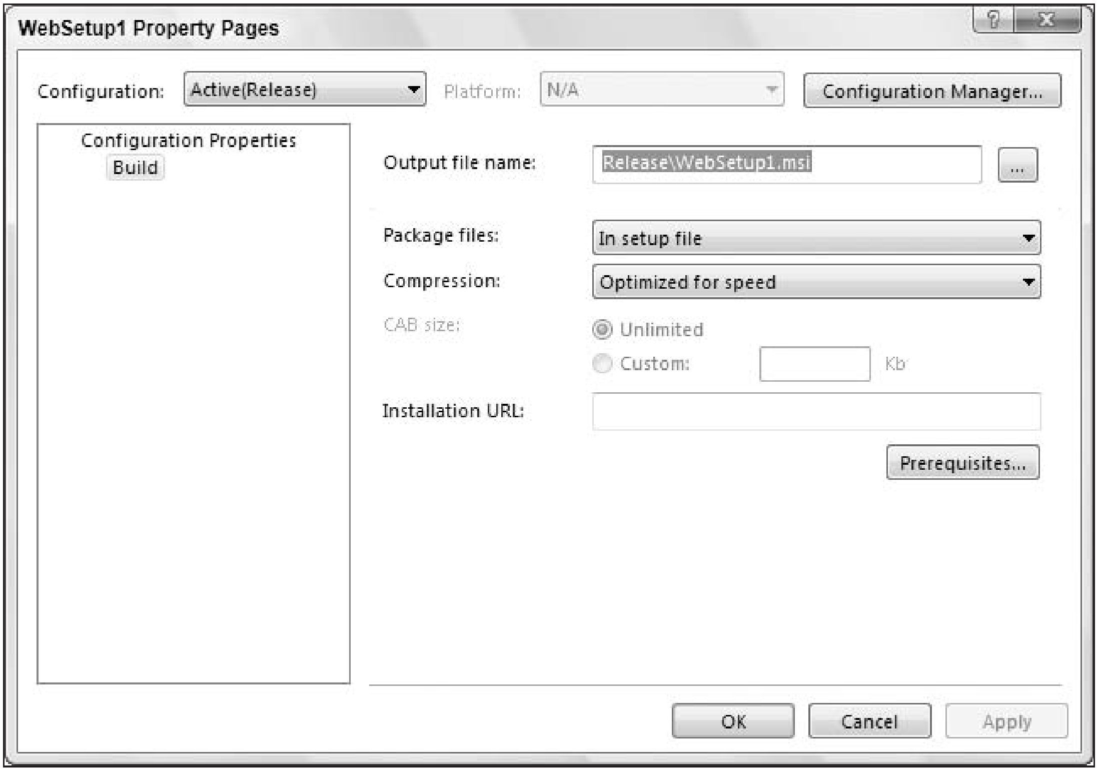
Working with the Deployment Project Properties

You can work with the project properties of the installer from Visual Studio in several ways. The first way is by right-clicking the installer project from the Solution Explorer of Visual Studio and selecting Properties from the menu. This pulls up the WebSetup1 Properties Pages dialog shown in Figure 34-22.

This dialog has some important settings for your installer application. Notice that, like other typical projects, this setup and deployment project allows for different active build configuration settings. For instance, you can have the active build configuration set to either Release or Debug. You can also click on the Configuration Manager button to get access to configuration settings for all the projects involved. In addition, this dialog enables you to add or remove build configurations from the project.

The Output File Name

The Output File Name setting lets you set the name of the `.msi` file that is generated. By default, it is the name of the project, but you can change this value to anything you want. This section also allows you to modify the location where the built `.msi` is placed on the system after the build process occurs.



Configuration: Active(Release)

Platform: N/A

Output file name: Release\WebSetup1.msi

Package files: In setup file

Compression: Optimized for speed

CAB size: ● Unlimited ○ Custom: Kb

Installation URL:

Figure 34-22

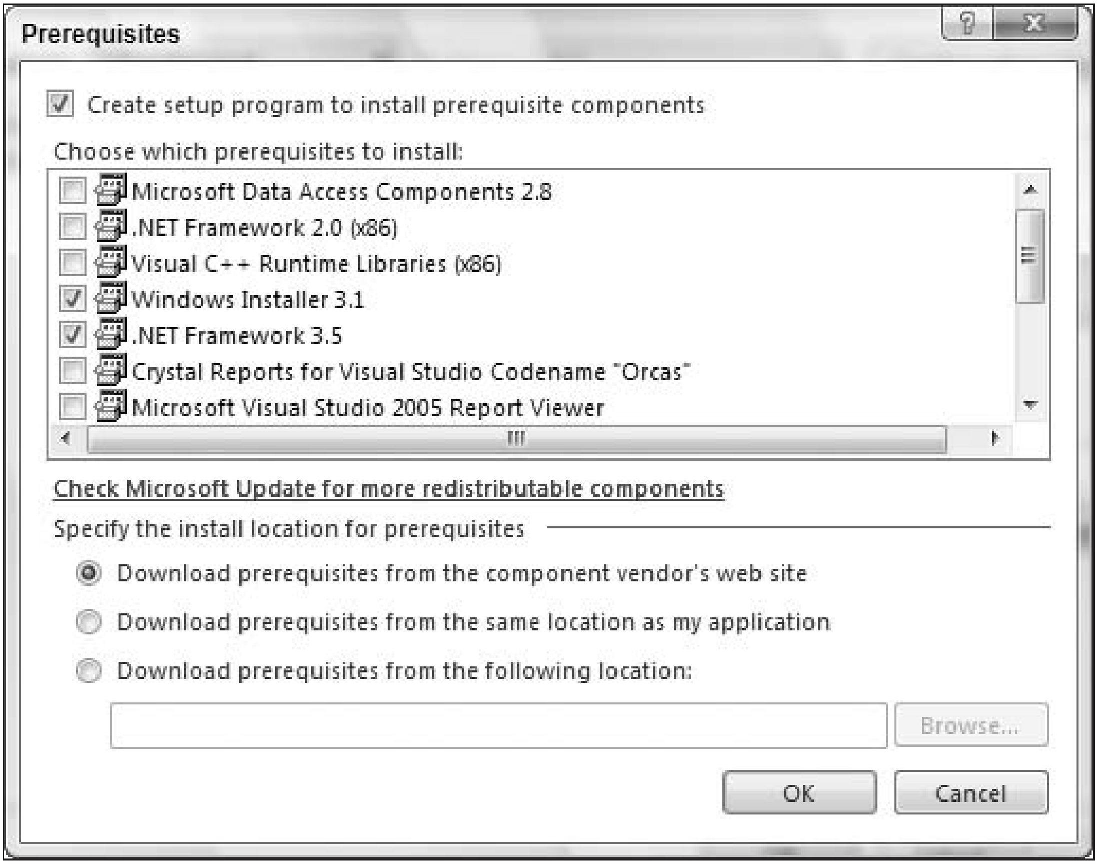
Package Files

The Package files section of this properties page enables you to specify how the application files are packaged in the `.msi` file. The available options include the following:

- ❑ **As loose, uncompressed files:** This option builds the project so that a resulting `.msi` file is created without the required application files. Instead, these application files are kept separate from the `.msi` file but copied to the same location as the `.msi` file. With this type of structure, you must distribute both the `.msi` file and the associated application files.
- ❑ **In setup file:** This option (which is the default option) packages the application files inside the `.msi` file. This makes distribution an easy task because only a single file is distributed.
- ❑ **In cabinet file(s):** This option packages all the application files into a number of cabinet files. The size of the cabinet files can be controlled through this same dialog (discussed shortly). This is an ideal type of installation process to use if you have to spread the installation application over a number of DVDs, CDs, or floppy disks.

Installation URL

Invariably, the ASP.NET applications you build have some component dependencies. In most cases, your application depends on some version of the .NET Framework. The installation of these dependencies, or components, can be made part of the overall installation process. This process is also referred to as *bootstrapping*. Clicking the Prerequisites button next to the Installation URL text box gives you a short list of available components that are built into Visual Studio in order to bootstrap to the installation program you are constructing (see Figure 34-23).



☑ Create setup program to install prerequisite components

Choose which prerequisites to install:

- ☐ Microsoft Data Access Components 2.8
- ☐ .NET Framework 2.0 (x86)
- ☐ Visual C++ Runtime Libraries (x86)
- ☑ Windows Installer 3.1
- ☑ .NET Framework 3.5
- ☐ Crystal Reports for Visual Studio Codename "Orcas"
- ☐ Microsoft Visual Studio 2005 Report Viewer

Check Microsoft Update for more redistributable components

Specify the install location for prerequisites

- ● Download prerequisites from the component vendor's web site
- ○ Download prerequisites from the same location as my application
- ○ Download prerequisites from the following location:

Figure 34-23

As you can see from when you first enter this settings dialog, the .NET Framework 3.5 and the Windows Installer 3.1 options are enabled by default, and you check the other components (thereby enabling them) only if your Web application has some kind of dependency on them.

From this dialog, you can also set how the dependent components are downloaded to the server where the installation is occurring. The options include downloading from Microsoft, from the server where the application originated, or from a defined location (URL) specified in the provided text box.

Compression

The Windows Installer service can work with the compression of the application files included in the build process so that they are optimized for either speed or size. You also have the option to turn off all compression optimizations. The default setting is Optimized for Speed.

CAB Size

The CAB Size section of the properties page is enabled only if you select In Cabinet File(s) from the Package Files drop-down list, as explained earlier. If this is selected, it is enabled with the `Unlimited` radio button selected. As you can see from this section, the two settings are `Unlimited` and `Custom`:

- ❑ `Unlimited`: This selection means that only a single cabinet file is created. The size of this file is dependent on the size of the collection of application files in the Web application and the type of compression selected.
- ❑ `Custom`: This selection allows you to break up the installation across multiple cabinet files. If the `Custom` radio button is selected, you can enter the maximum size of the cabinet files allowed in the provided text box. The measure of the number you place in the text box is in kilobytes (KB).

Additional Properties

You learned one place where you can apply settings to the installer program; however, at another place in Visual Studio you can find even more properties pertaining to the entire installer program. By selecting the `WebSetup1` installer program in the Solution Explorer, you can work with the installer properties directly from the Properties window of Visual Studio. The following table lists the properties that appear in the Properties window.

Property	**Description**
`AddRemoveProgramsIcon`	Defines the location of the icon used in the Add/Remove Programs dialog found through the system's Control Panel.
`Author`	The author of the installer. This could be the name of a company or individual.
`Description`	Allows for a textual description of the installer program.
`DetectNewerInstalledVersion`	Instructs the installer to make a check on the installation server if a newer version of the application is present. If one is present, the installation is aborted. The default setting is `True` (meaning that the check will be made).
`Keywords`	Defines the keywords used when a search is made for an installer.
`Localization`	Defines the locale for any string resources and the runtime user interface. An example setting is `English (United States)`.
`Manufacturer`	Defines the name of the company that built or provided the installer program.
`ManufacturerUrl`	Defines the URL of the company that built or provided the installer program.
`PostBuildEvent`	Specifies a command line executed after the build ends.
`PreBuildEvent`	Specifies a command line executed before the build begins.

Property	**Description**
`ProductCode`	Defines a string value that is the unique identifier for the application. An example value is `{885D2E86-6247-4624-9DB1-50790E3856B4}`.
`ProductName`	Defines the name of the program being installed.
`RemovePreviousVersions`	Specifies as a `Boolean` value whether any previous versions of the application should be uninstalled prior to installing the fresh version. The default setting is `False`.
`RestartWWWService`	Specifies as a `Boolean` value whether or not IIS should be stopped and restarted for the installation process. The default value is `False`.
`RunPostBuildEvent`	Defines when to run the post-build event. The default setting is `On successful build`. The other possible value is `Always`.
`SearchPath`	Defines the path to use to search for any files, assemblies, merge modules on the development machine.
`Subject`	Allows you to provide additional descriptions for the application.
`SupportPhone`	Specifies the support telephone number for the installed program.
`SupportUrl`	Specifies the URL by which the end user can get support for the installed application.
`TargetPlatform`	Defines the target platform of the installer. Possible values include `x86`, `x64`, and `Itanium`.
`Title`	Defines the title of the installer program.
`UpgradeCode`	Defines a shared identifier that can be used from build to build. An example value is `{A71833C7-3B76-4083-9D34-F074A4FFF544}`.
`Version`	Specifies the version number of the installer, cabinet file, or merge module. An example value is `1.0.1`.

The following sections look at the various editors provided to help you build and customize the construction of the installer. You can get at these editors by clicking the appropriate icon in the Solution Explorer in Visual Studio or by choosing View ➪ Editor in the Visual Studio menu. These editors are explained next.

The File System Editor

The first editor that comes up when you create your installer program is the File System Editor. The File System Editor enables you to add folders and files that are to be installed on the destination server. In addition to installing folders and files, it also facilitates the creation of shortcuts. This editor is shown in Figure 34-24.

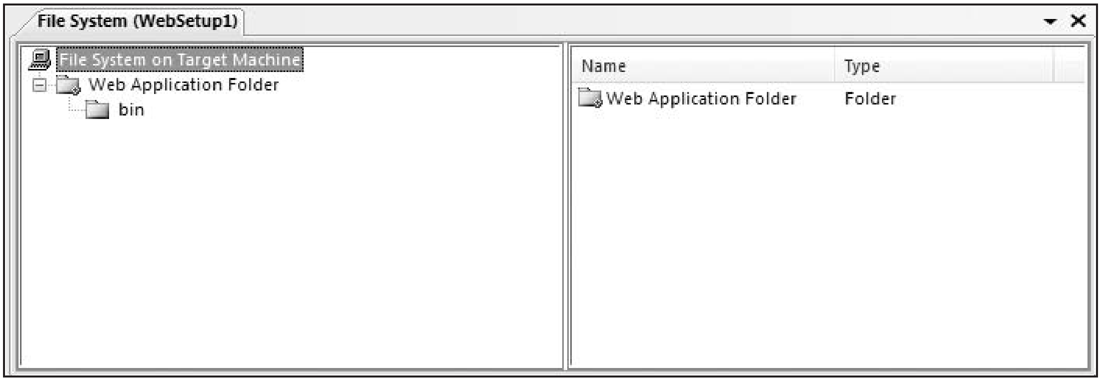


Figure 34-24

The File System Editor has two sections. The left section is the list of folders to be installed during the installation process. By default, only the Web Application Folder is shown. Highlighting this folder, or one of the other folders, gives you a list of properties for that folder in the Properties window of Visual Studio. The following table details some of the properties you might find in the Properties window.

Property	Description
`AllowDirectoryBrowsing`	Allows browsing of the selected directory in IIS. The default value is `False`.
`AllowReadAccess`	Specifies whether the selected folder should have Read access. The default value is `True`.
`AllowScriptSourceAccess`	Specifies the script source access of the selected folder. The default value is `False`.
`AllowWriteAccess`	Specifies whether the selected folder should have Write access. The default value is `False`.
`ApplicationProtection`	Defines the IIS Application Protection property for the selected folder. Possible values include `vsdapLow`, `vsdapMedium`, and `vsdapHigh`. The default value is `vsdapMedium`.
`AppMappings`	Enables you to define the IIS application mappings for the selected folder.
`DefaultDocument`	Defines the default document of the selected folder. The default value is `Default.aspx`.
`ExecutePermissions`	Defines the IIS Execute Permissions property. Possible values include `vsdepNone`, `vsdepScriptsOnly`, `vsdepScriptsAndExecutables`. The default value is `vsdepScriptsOnly`.
`Index`	Specifies the IIS Index of this resource property for the selected folder. The default value is `True`.
`IsApplication`	Specifies whether an IIS application root is created for the installed application. The default value is `True`.

Property	Description
`LogVisits`	Specifies the IIS Log Visits property for the selected folder. The default value is `True`.
`VirtualDirectory`	Defines the name of the virtual directory created. The default value is the name of the project.

Adding Items to the Output

You can add files, folders, and assemblies to the installer output quite easily. To add some of these items to the output list, right-click the folder and select Add from the menu. You have four choices: Web Folder, Project Output, File, and Assembly.

If you want to add a custom folder to the output (for example, an Images folder), you can select Web Folder and provide the name of the folder. This enables you to create the folder structure you want.

If you want to add system folders, you highlight the File System on Target Machine node and then choose Action ➪ Add Special Folder. This provides you with a large list of folders that are available for you to add to the installer program. You can also get at this list of folders by simply right-clicking a blank portion of the left pane of the File System Editor (see Figure 34-25).

The following table defines the possible folders you can add to the installer structure you are building.

Folders and Menus	Description
Common Files Folder	Meant for non–system files not shared by multiple applications.
Common Files (64-bit) Folder	Meant for non–system files on a 64-bit machine not shared by multiple applications.
Fonts Folder	Meant for only fonts you want installed on the client's machine.
Program Files Folder	A Windows Forms application would be a heavy user of this folder because most applications are installed here.
Program Files (64-bit) Folder	A Programs Files folder meant for 64-bit machines.
System Folder	Meant for storing files considered shared system files.
System (64-bit) Folder	Meant for storing files on 64-bit machines considered shared system files.
User's Application Data Folder	A hidden folder meant for storing data that is application- and user-specific.
User's Desktop	Meant for storing files on a user's desktop (also stores these files in the My Desktop folder).
User's Favorites Folder	Meant for storing files in a user's Favorites folder (browser-specific).

Folders and Menus	Description
User's Personal Data Folder	Meant for storing personal data specific to a single user. This is also referred to as the My Documents folder.
User's Programs Menu	Meant for storing shortcuts, which then appear in the user's program menu.
User's Send To Menu	Meant for storing files that are presented when a user attempts to send a file or folder to a specific application (by right-clicking the folder or file and selecting Send To).
User's Start Menu	Meant for storing files in the user's Start menu.
User's Startup Folder	Meant for storing files that are initiated whenever a user logs into his machine.
User's Template Folder	Meant for storing templates (applications like Microsoft's Office).
Windows Folder	Meant for storing files in the Windows root folder. These are usually system files.
Global Assembly Cache Folder	Meant for storing assemblies that can then be utilized by all the applications on the server (shared assemblies).
Custom Folder	Another way of creating a unique folder.
Web Custom Folder	Another way of creating a unique folder that also contains a bin folder.

Creating a Desktop Shortcut to the Web Application

For an example of using one of these custom folders, look at placing a shortcut to the Web application on the user's desktop. The first step is to right-click on a blank portion of the left-hand pane in the File System Editor and choose Add Special Folder ➪ User's Desktop. This adds that folder to the list of folders presented in the left-hand pane.

Because you want to create a desktop shortcut to the Web Application Folder and not to the desktop itself, the next step is to right-click the Web Application folder and select Create Shortcut to Web Application Folder. The created shortcut appears in the right-hand pane. Right-click the shortcut and rename it to something a little more meaningful, such as Wrox Application. Because you do not want to keep the shortcut in this folder, drag the shortcut from the Web Application Folder and drop it onto the User's Desktop folder.

With this structure in place, this installer program not only installs the application (as was done previously), but it also installs the application's shortcut on the user's desktop.

The Registry Editor

The next editor is the Registry Editor. This editor enables you to work with the client's registry in an easy and straightforward manner. Using this editor, you can perform operations such as creating new registry keys, providing values for already existing registry keys, and importing registry files. The Registry Editor is presented in Figure 34-26.

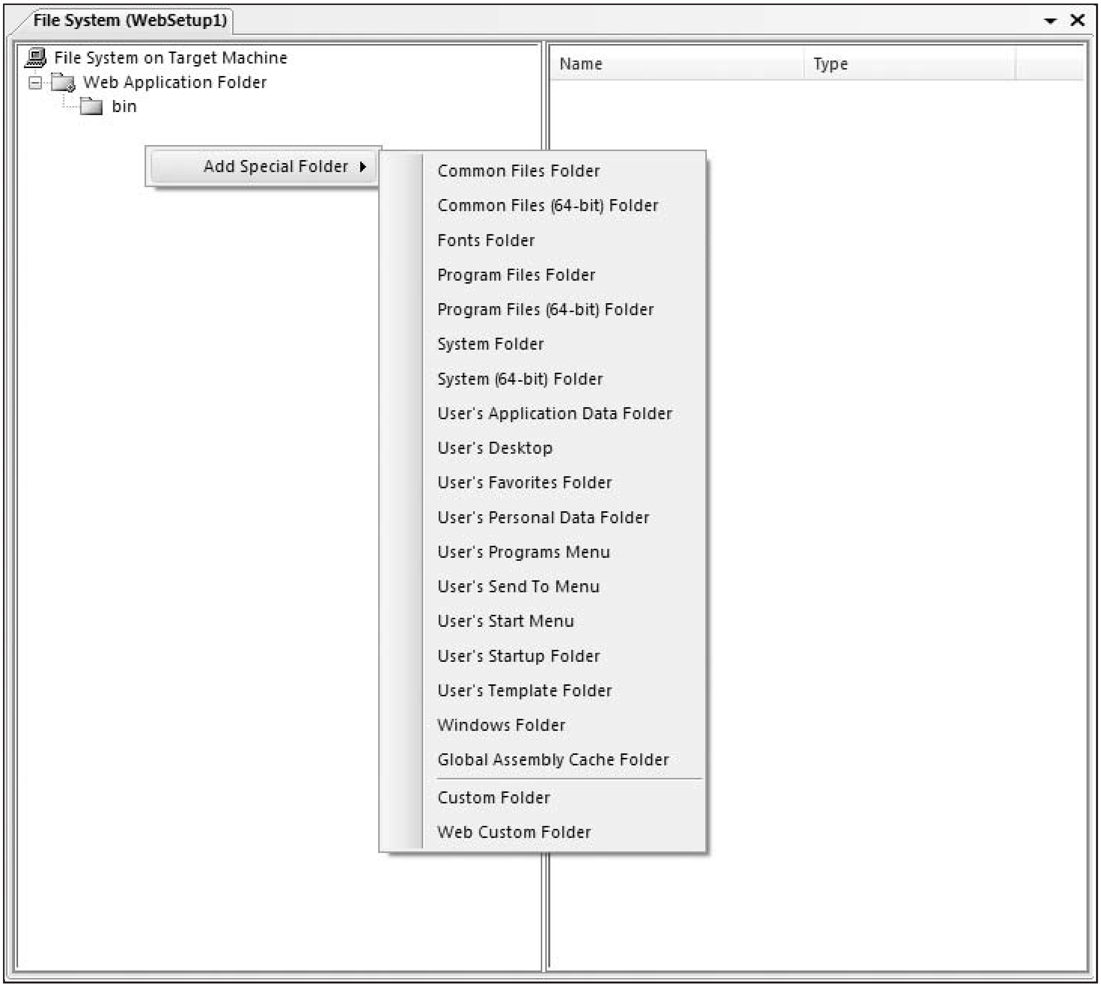

File System (WebSetup1)
File System on Target Machine
Web Application Folder
bin
Name
Type
Add Special Folder
Common Files Folder
Common Files (64-bit) Folder
Fonts Folder
Program Files Folder
Program Files (64-bit) Folder
System Folder
System (64-bit) Folder
User's Application Data Folder
User's Desktop
User's Favorites Folder
User's Personal Data Folder
User's Programs Menu
User's Send To Menu
User's Start Menu
User's Startup Folder
User's Template Folder
Windows Folder
Global Assembly Cache Folder
Custom Folder
Web Custom Folder

Figure 34-25

From this figure, you can see that the left-hand pane provides the standard registry folders, such as HKEY_CLASSES_ROOT and HKEY_LOCAL_MACHINE, as well as others. Right-clicking one of these folders, you can add a new key from the menu selection. This creates a new folder in the left-hand pane where it is enabled for renaming. By right-clicking this folder, you can add items such as those illustrated in Figure 34-27.

As you can see in the figure, you can add items such as the following:

- Key
- String Value

- Environment String Value
- Binary Value
- DWORD Value

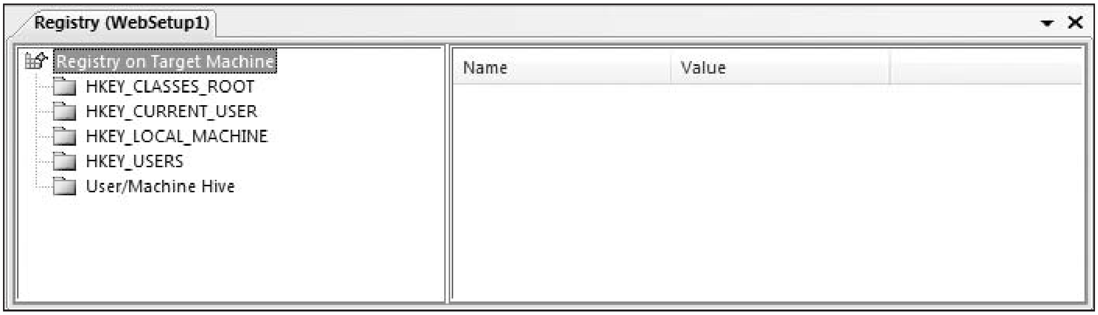

Registry (WebSetup1)
Registry on Target Machine
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
User/Machine Hive
Name
Value

Figure 34-26

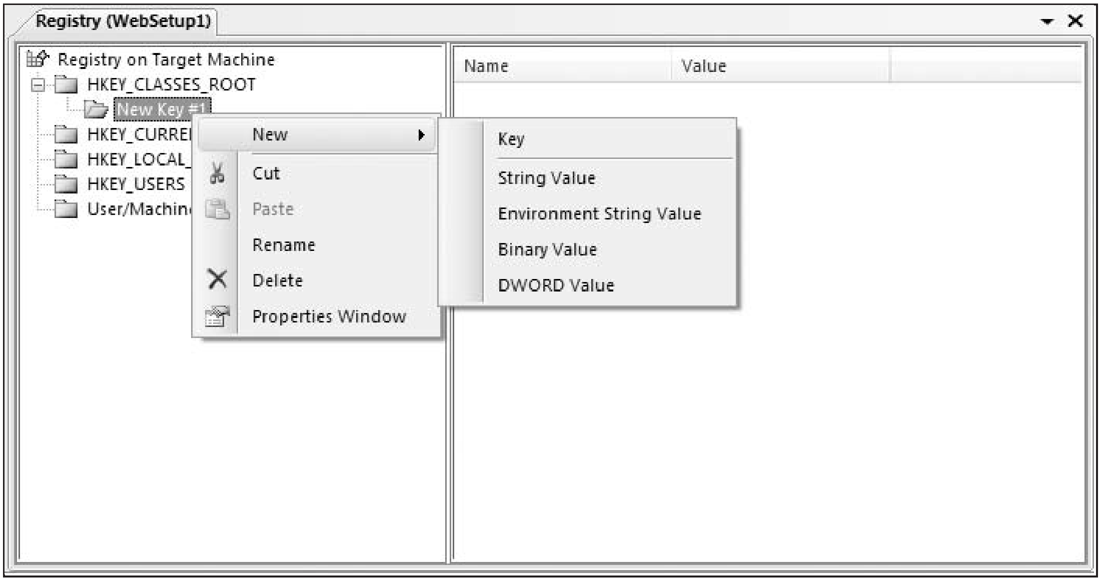

Registry (WebSetup1)
Registry on Target Machine
HKEY_CLASSES_ROOT
New Key #1
HKEY_CURREI
HKEY_LOCAL_
HKEY_USERS
User/Machin
New
Cut
Paste
Rename
Delete
Properties Window
Key
String Value
Environment String Value
Binary Value
DWORD Value
Name
Value

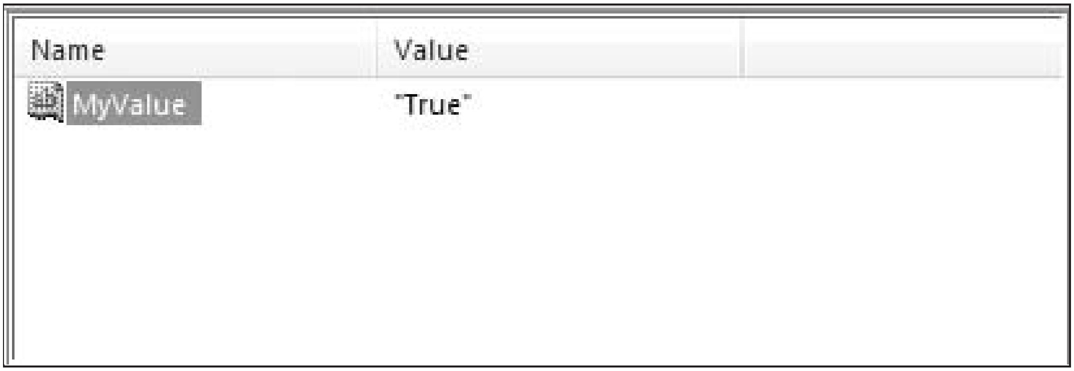
Figure 34-27

Selecting String Value allows you to apply your settings for this in the right-hand pane, as illustrated in Figure 34-28.

The other values work in a similar manner.

The File Types Editor

All files on a Windows operating system use file extensions to uniquely identify themselves. A file such as `Default.aspx`, for example, uses the file extension `.aspx`. This file extension is then associated with



Name	Value
MyValue	"True"

Figure 34-28

ASP.NET. Another example is `.xls`. This file extension is associated with Microsoft Excel. When someone attempts to open an `.xls` file, the file is passed to the Excel program because of mappings that have been made on the computer to associate these two entities.

Using the File Types Editor in Visual Studio, you can also make these mappings for the applications you are trying to install. Right-clicking the File Types On Target Machine allows you to add a new file type. From here, you can give your file type a descriptive name and provide a file extension (shown in Figure 34-29).

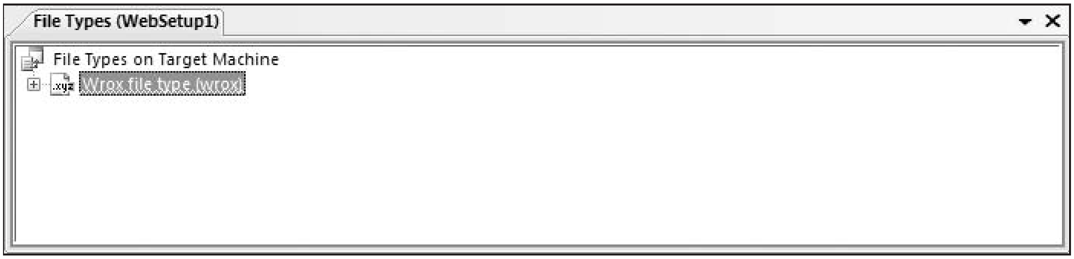


Figure 34-29

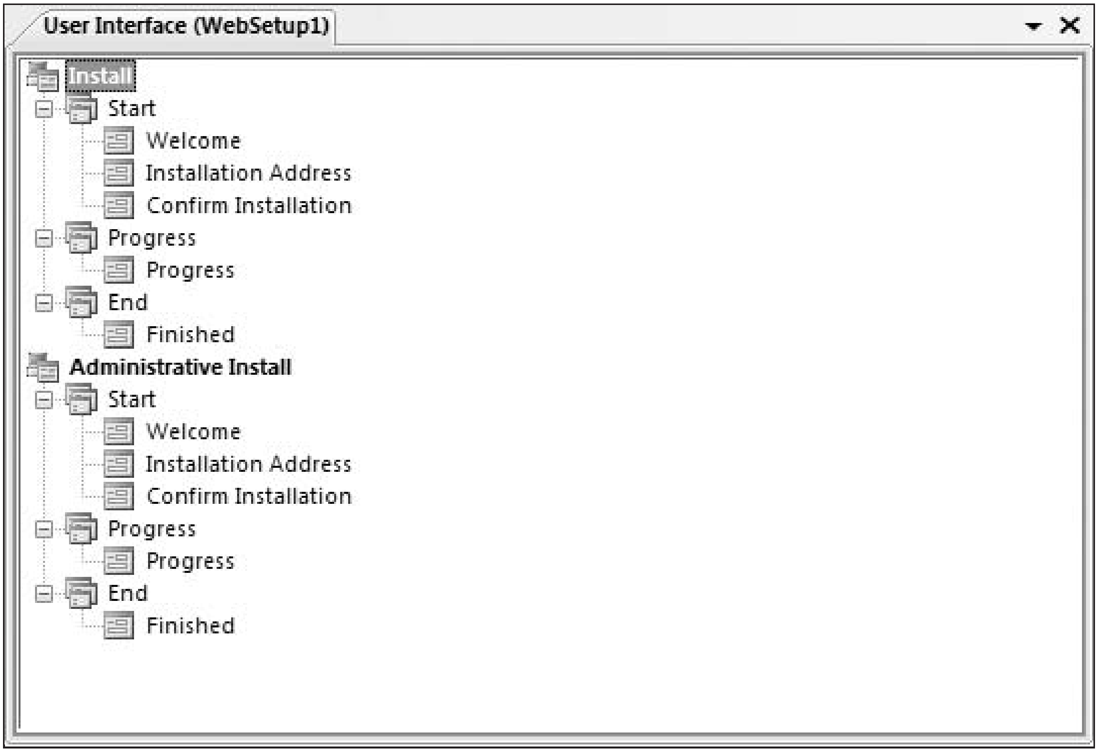
Highlighting the defined file type provides some properties that you can set in the Visual Studio Properties window, as shown in the following table.

Property	**Description**
`Name`	Specifies a name used in the File System Editor to identify a file type and its associated settings.
`Command`	Specifies the executable file (`.exe`) that is launched when the specified file extension is encountered.
`Description`	Defines a textual description for the file type.

Property	Description
`Extensions`	Defines the file extension associated with the executable through the `Command` property. An example is `wrox`. You should specify the extension without the period in front of it. If you are going to specify multiple extensions, you can provide a list separated by semicolons.
`Icon`	Defines the icon used for this file extension.
`MIME`	Specifies the MIME type associated with this file type. An example is `application/msword`.

The User Interface Editor

The User Interface Editor defines the dialogs used in the installation process. You can change the installation process greatly with the dialogs you decide to use or not use. By default, these dialogs (shown in Figure 34-30) are presented in your installer.



User Interface (WebSetup1)

- Install
  - Start
    - Welcome
    - Installation Address
    - Confirm Installation
  - Progress
    - Progress
  - End
    - Finished
- Administrative Install
  - Start
    - Welcome
    - Installation Address
    - Confirm Installation
  - Progress
    - Progress
  - End
    - Finished

Figure 34-30

From this figure, you can see how the dialogs are divided into two main sections. The first section, labeled Install, is the dialog sequence used for a typical install. However, because some applications might require it, a second installation process is defined through the Administrative Install. The Administrative Install process is initiated only if the user is logged onto the machine under the Administrator account. If this is not the case, the Install section is used instead.

By default, the Install and Administrative Install sections are exactly the same. Both the Install and Administrative Install sections are further divided into three subsections: Start, Progress, and End. These sections are defined in the following list:

- ❑ **Start:** A sequence of dialogs that appears before the installation occurs. By default, the Start section includes a welcome screen, a definition stating where the application is to be installed, and a dialog asking for an installation confirmation.
- ❑ **Progress:** The second stage, the Progress stage, is the stage in which the actual installation occurs. Throughout this stage no interaction occurs between the installer program and the end user. This is the stage where the end user can watch the installation progress through a progress bar.
- ❑ **End:** The End stage specifies to the end user whether the installation was successful. Many installer programs use this stage to present the customer with release notes and `ReadMe.txt` files, as well as the capability to launch the installed program directly from the installer program itself.

Adding Dialogs to the Installation Process

Of course, you are not limited to just the dialogs that appear in the User Interface Editor by default. You have a number of other dialogs that can be added to the installation process. For instance, right-click the Start node and select Add Dialog (or highlight the Start node and choose Action ➪ Add Dialog). This pulls up the Add Dialog dialog, as shown in Figure 34-31.

As you can see from this image, you can add quite a number of different steps to the installation process, such as license agreements and splash screens. After adding a dialog to the process, you can highlight the dialog to get its properties to appear in the Properties window so that you can assign the items needed. For example, you can assign the image to use for the splash screen or the `.rtf` file to use for the license agreement.

When you add an additional dialog to the installation process (for instance, to the Install section), be sure to also install the same dialog on the Administrative Install (if required). If no difference exists between the two user types in the install process, be sure to add the dialogs in unison in order to keep them the same.

Changing the Order in Which the Dialogs Appear in the Process

In working with the dialogs in the Start, Process, and End sections of the User Interface Editor, you can always determine the order in which these dialogs appear. Even if you are working with the default dialogs, you can easily change their order by right-clicking the dialog and selecting Move Up or Move Down, as shown in Figure 34-32.

The Custom Actions Editor

The Custom Actions Editor is a powerful editor that enables you to take the installer one step further and perform custom actions during various events of the installation cycle (but always *after* the installation

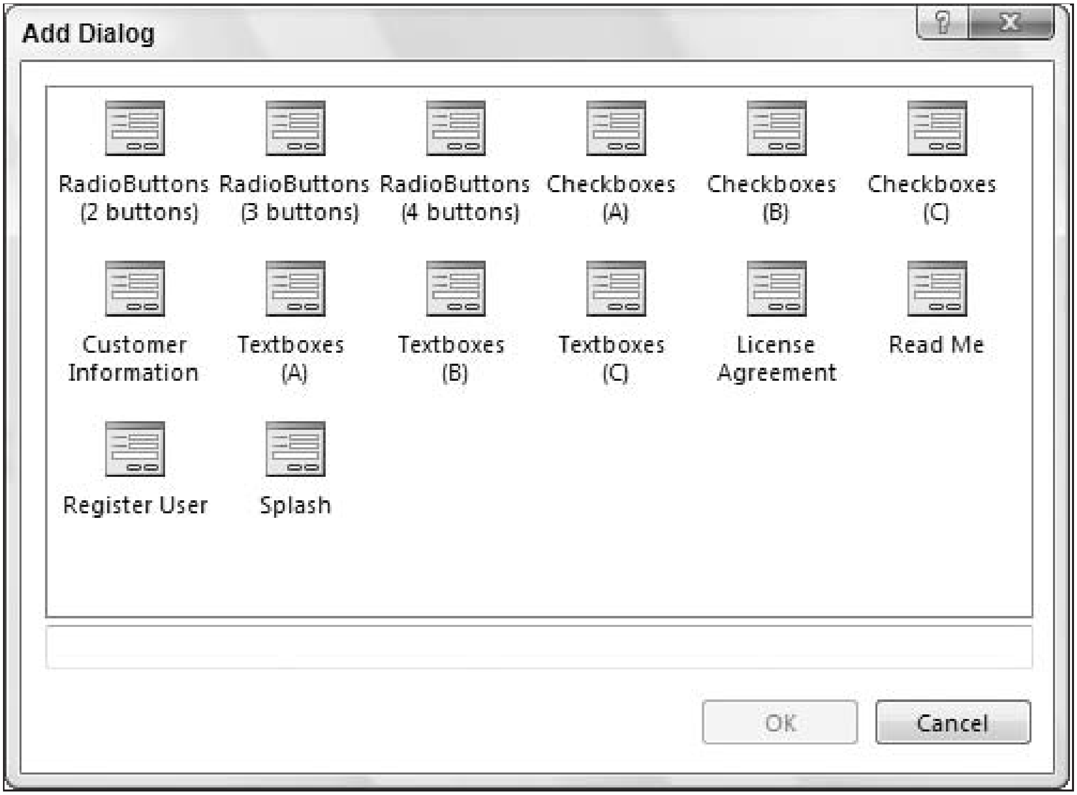

Add Dialog
RadioButtons (2 buttons)
RadioButtons (3 buttons)
RadioButtons (4 buttons)
Checkboxes (A)
Checkboxes (B)
Checkboxes (C)
Customer Information
Textboxes (A)
Textboxes (B)
Textboxes (C)
License Agreement
Read Me
Register User
Splash
OK
Cancel

Figure 34-31

process is completed) such as Install, Commit, Rollback, and Uninstall. The Custom Actions Editor is presented in Figure 34-33.

The idea is that you can place a reference to a `.dll`, `.exe`, or `.vbs` file from one of the folders presented here in the Custom Actions Editor to perform a custom action. For example, you can insert a custom action to install a database into Microsoft's SQL Server in the Commit folder (after the install has actually been committed).

The four available folders are explained in the following list:

- **Install:** This is the point at which the installation of the files for the Web application are finished being installed. Although the files are installed, this point is right before the installation has been committed.
- **Commit:** This is the point at which the actions of the installation have been actually committed (taken) and are considered successful.
- **Rollback:** This is the point at which the installation has failed and the computer must return to the same state that it was in before the installation occurred.
- **Uninstall:** This is the point at which a successfully installed application is uninstalled for a machine.

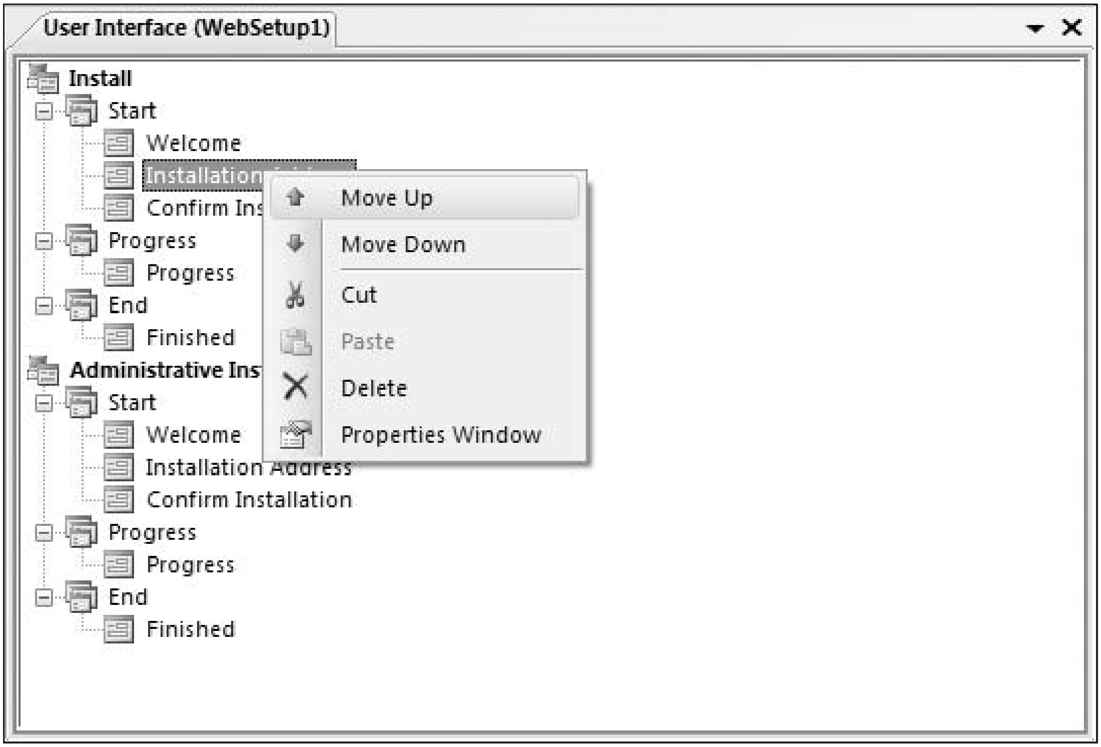

User Interface (WebSetup1)
Install
Start
Welcome
Installation
Confirm Ins
Progress
Progress
End
Finished
Administrative Ins
Start
Welcome
Installation Address
Confirm Installation
Progress
Progress
End
Finished
Move Up
Move Down
Cut
Paste
Delete
Properties Window

Figure 34-32

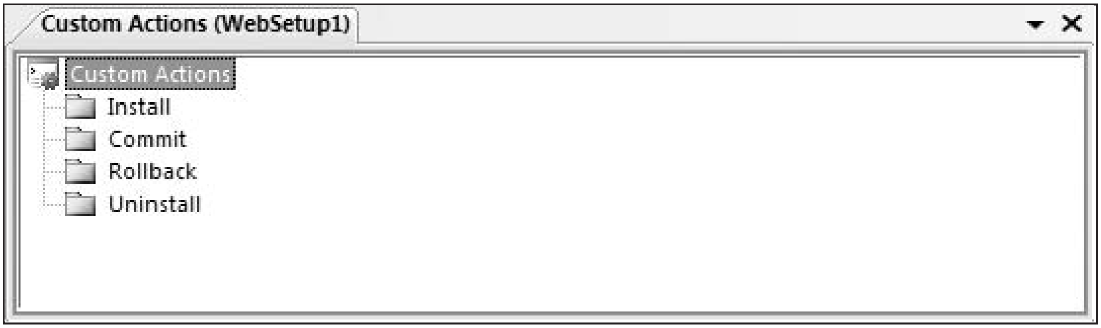

Custom Actions (WebSetup1)
Custom Actions
Install
Commit
Rollback
Uninstall

Figure 34-33

Using these capabilities, you can take the installation process to the level of complexity you need for a successfully installed application.

The Launch Conditions Editor

Certain conditions are required in order for your Web application to run on another server automatically. Unless your application is made up of HTML files only, you must make sure that the .NET Framework is installed on the targeted machine in order to consider the install a success. The Launch Conditions Editor is an editor that you can use to make sure that everything that needs to be in place on the installation computer for the installation to occur is there. The Launch Conditions Editor is presented in Figure 34-34.

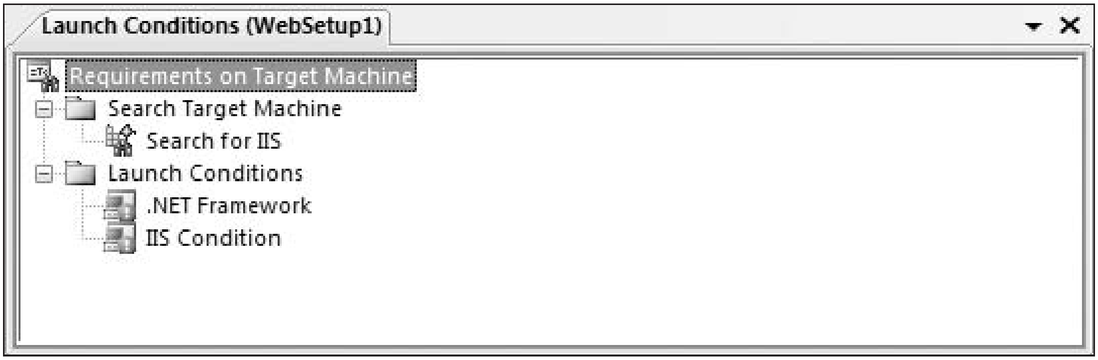


Figure 34-34

From this image, you can see some of the conditions required in this instance. The first folder defines the items that must be in place on the computer where the installation is to occur. A search is done on the computer to see whether IIS is installed. It can also check if any files or registry keys are present on the computer before the installation occurs.

The second folder is an important one because certain conditions must be in place before the installation. This folder shows two conditions. One is that the .NET Framework must be installed, and the second is that IIS must be installed. You can add these types of launch conditions by right-clicking the Requirements On Target Machine node in the dialog. You are then presented with a short list of conditions.

After a condition is in place, you can highlight the condition to see the property details of this condition in the Properties window. For instance, highlighting the IIS Condition gives you some basic properties in the Properties window. One of these is the `Condition` property. By default, for an IIS Condition, the value of the `Condition` property is the following:

```
IISVERSION >= "#4"
```

This means that the requirement for this installation is that IIS must be equal to or greater than version 4. If it is not, the installation fails. If the IIS version is 4, 5, or 6, the installation can proceed. You can feel free to change this value to whatever you deem necessary. You can change the value to `IISVERSION >= "#5"`, for example, to ensure it is either IIS 5.0, 6.0, or 7.0 at a minimum.

Another example of fine-tuning these launch conditions is the .NET Framework condition that enables you to set the minimum version of the .NET Framework you want to allow. You do this by setting the `Version` property of the condition.

Summary

As you can see, you have many possibilities for installing your ASP.NET applications! From the simplest mode of just copying the files to a remote server — sort of a save-and-run mode — to building a complex installer program that can run side events, provide dialogs, and even install extra items such as databases and more.

Just remember that when working on the installation procedures for your Web applications, you should be thinking about making the entire process logical and easy for your customers to understand. You do not want to make people’s lives too difficult when they are required to programmatically install items on another machine.

A

Migrating Older ASP.NET Projects

In some cases, you build your ASP.NET 3.5 applications from scratch — starting everything new. In many instances, however, this is not an option. You need to take an ASP.NET application that was previously built on the 1.0, 1.1, or 2.0 versions of the .NET Framework and migrate the application so that it can run on the .NET Framework 3.5.

This appendix focuses on migrating ASP.NET 1.*x* or 2.0 applications to the 3.5 framework.

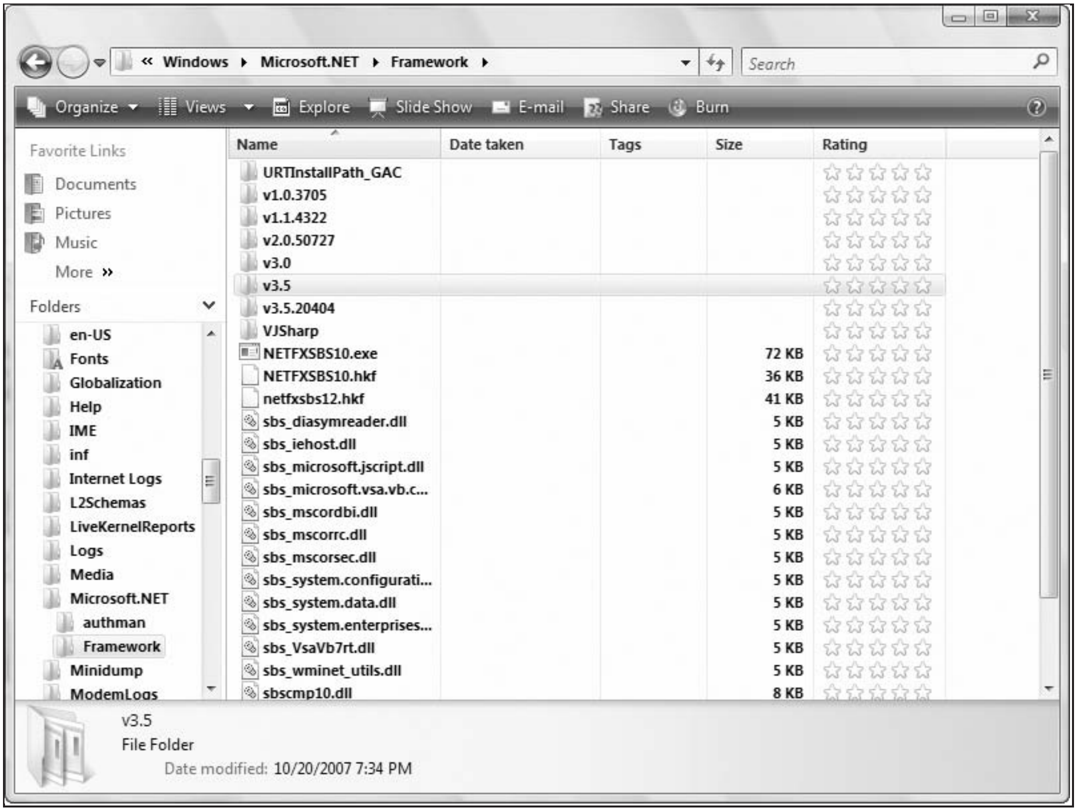
Migrating Is Not Difficult

Be aware that Microsoft has done a lot of work to ensure that the migration process from ASP.NET 1.*x* is as painless as possible. In most cases, your applications run with no changes needed.

When moving a 1.*x* or 2.0 application to 3.5, you don't have to put the ASP.NET application on a new server or make any changes to your present server beyond installing the .NET Framework 3.5.

After you install the .NET Framework 3.5, you see the framework versions on your server at C:\WINDOWS\Microsoft.NET\Framework, as illustrated in Figure A-1.

In this case, you can see that all five official versions of the .NET Framework installed including v1.0.3705, v1.1.4322, v2.0.50727, v3.0, and v3.5.


« Windows ▸ Microsoft.NET ▸ Framework ▸
Search
Organize
Views
Explore
Slide Show
E-mail
Share
Burn
Favorite Links
Documents
Pictures
Music
More »
Folders
en-US
Fonts
Globalization
Help
IME
inf
Internet Logs
L2Schemas
LiveKernelReports
Logs
Media
Microsoft.NET
authman
Framework
Minidump
ModemLogs
v3.5
File Folder
Date modified: 10/20/2007 7:34 PM

Name	Date taken	Tags	Size	Rating
URTInstallPath_GAC				
v1.0.3705				
v1.1.4322				
v2.0.50727				
v3.0				
v3.5				
v3.5.20404				
VJSharp				
NETFXSBS10.exe			72 KB	
NETFXSBS10.hkf			36 KB	
netfxsbs12.hkf			41 KB	
sbs_diasymreader.dll			5 KB	
sbs_iehost.dll			5 KB	
sbs_microsoft.jscript.dll			5 KB	
sbs_microsoft.vsa.vb.c...			6 KB	
sbs_mscordbi.dll			5 KB	
sbs_mscorrc.dll			5 KB	
sbs_mscorsec.dll			5 KB	
sbs_system.configurati...			5 KB	
sbs_system.data.dll			5 KB	
sbs_system.enterprises...			5 KB	
sbs_VsaVb7rt.dll			5 KB	
sbs_wminet_utils.dll			5 KB	
sbscmp10.dll			8 KB	

Figure A-1

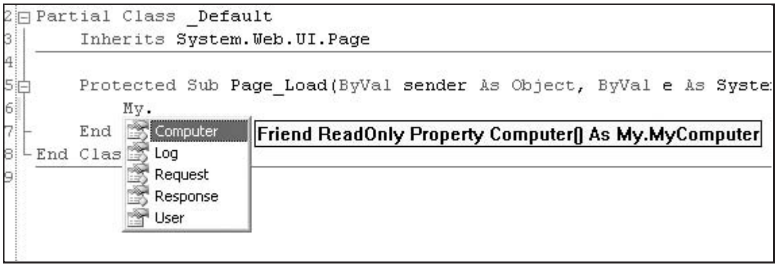
Running Multiple Versions of the Framework Side by Side

From this figure, you can see that it is possible to run multiple versions of the .NET Framework side by side. ASP.NET 1.0, ASP.NET 1.1, ASP.NET 2.0, and ASP.NET 3.5 applications can all run from the same server. Different versions of ASP.NET applications that are running on the same server run in their own worker processes and are isolated from one another.

Upgrading Your ASP.NET Applications

When you install the .NET Framework 3.5, it does not remap all your ASP.NET applications so that they now run off of the new framework instance. Instead, you selectively remap applications to run off of the ASP.NET 3.5 framework.

To accomplish this task if you are migrating ASP.NET 1.*x* applications to ASP.NET 2.0, you use the ASP.NET MMC Snap-In that is a part of the .NET Framework 2.0 install. You get to this GUI-based administration application by right-clicking and selecting Properties from the provided menu using Windows XP when you are working with your application domain in Microsoft's Internet Information Services (IIS). After selecting the MMC console (the Properties option), select the ASP.NET tab and you are provided with something similar to what is shown in Figure A-2.



```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As Syste
        My.
    End
End Clas
```

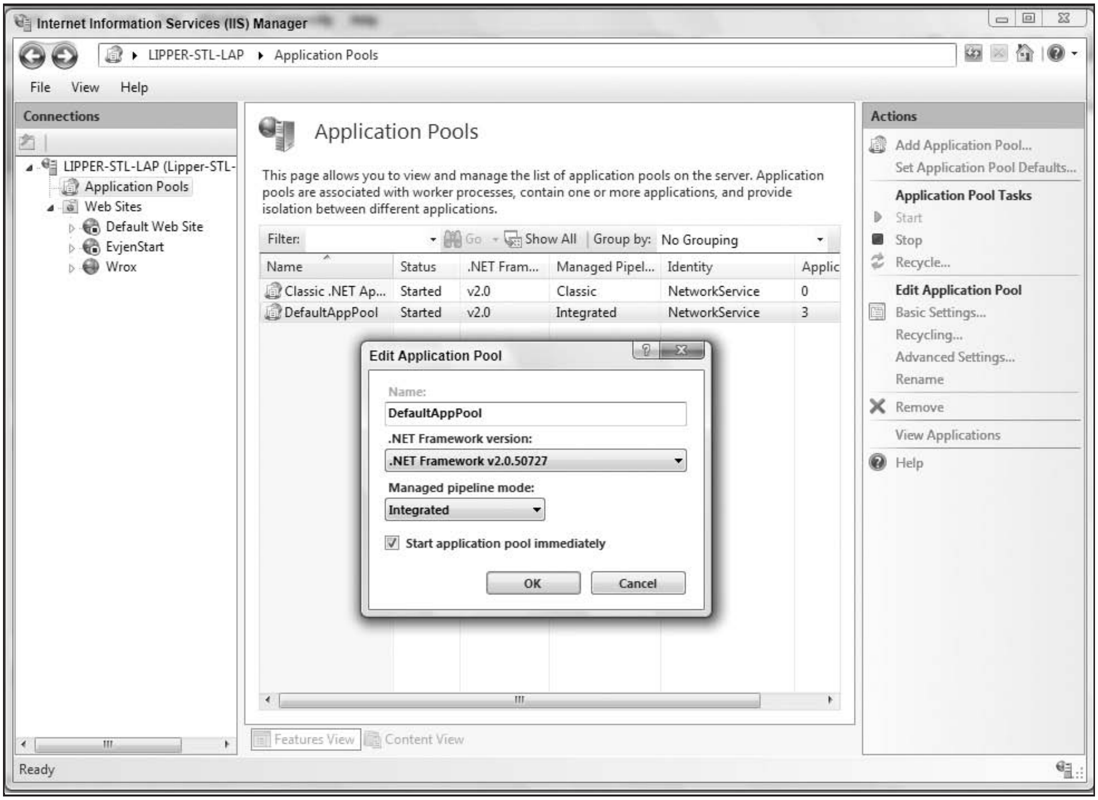
Computer
Log
Request
Response
User

Friend ReadOnly Property Computer() As My.MyComputer

Figure A-2

You can see from this figure that an option exists for selecting the application's ASP.NET version (the top-most option). This allows you to select the version of the .NET Framework in which this ASP.NET application should run. In this case, the Wrox application on my machine was retargeted to the 2.0 release of ASP.NET when I selected 2.0.50727 in the drop-down list.

> *You should always test your older ASP.NET application by first running on the newer version of ASP.NET in a developer or staging environment. Do not change the version to a newer version on a production system without first testing for any failures.*



Internet Information Services (IIS) Manager
LIPPER-STL-LAP ▸ Application Pools
File
View
Help
Connections
LIPPER-STL-LAP (Lipper-STL-
Application Pools
Web Sites
Default Web Site
EvjenStart
Wrox
Application Pools
Filter:
Go
Show All
Group by:
No Grouping
Actions
Add Application Pool...
Set Application Pool Defaults...
Application Pool Tasks
Start
Stop
Recycle...
Edit Application Pool
Basic Settings...
Recycling...
Advanced Settings...
Rename
Remove
View Applications
Help
Features View
Content View
Ready

This page allows you to view and manage the list of application pools on the server. Application pools are associated with worker processes, contain one or more applications, and provide isolation between different applications.

Name	Status	.NET Fram...	Managed Pipel...	Identity	Applic
Classic .NET Ap...	Started	v2.0	Classic	NetworkService	0
DefaultAppPool	Started	v2.0	Integrated	NetworkService	3

Edit Application Pool

Name: DefaultAppPool

.NET Framework version: .NET Framework v2.0.50727

Managed pipeline mode: Integrated

☑ Start application pool immediately

OK Cancel

Figure A-3

If you are not ready to upgrade your entire application to a newer version of ASP.NET, one option is to create additional virtual directories in the root virtual directory of your application and target the portions of the application to the versions of the .NET Framework that you want them to run on. This enables you to take a stepped approach in your upgrade process.

If you are upgrading from ASP.NET 2.0 to ASP.NET 3.5, there really is very little that you have to do. It is true that the System.Web DLL in both versions of the framework is the same. The major differences between these two versions of the framework for ASP.NET are the use of a different language compiler and the inclusion of the `System.Core` and the `System.Web.Extensions` DLLs.

The differences are even more self-evident when working with the new IIS Manager on Windows Vista. From this management tool, you can see that the DefaultAppPool is running off the same version 2.0.50727 of the .NET Framework, as shown in Figure A-3.

Upgrading your application to ASP.NET 3.5 using Visual Studio 2008 will cause the IDE to make all the necessary changes to the application's configuration file. This is illustrated later in this appendix.

When Mixing Versions — Forms Authentication

If you have an ASP.NET application that utilizes multiple versions of the .NET Framework then, as was previously mentioned, you must be aware of how forms authentication works in ASP.NET 2.0 and 3.5.

In ASP.NET 1.*x*, the forms authentication process uses Triple DES encryption (3DES) for the encryption and decryption process of the authentication cookies. ASP.NET 2.0 and 3.5, on the other hand, uses an encryption technique — AES (*Advanced Encryption Standard*).

AES is faster and more secure. However, because the two encryption techniques are different, you must change how ASP.NET 3.5 generates these keys. You can accomplish this by changing the `<machineKey>` section of the `web.config` in your ASP.NET 3.5 application so that it works with Triple DES encryption instead (as presented in Listing A-1).

Listing A-1: Changing your ASP.NET 3.5 application to use Triple DES encryption

```
<configuration>
   <system.web>

      <machineKey validation="3DES" decryption="3DES"
       validationKey="1234567890123456789012345678901234567890"
       decryptionKey="1234567890123456789012345678901234567890" />

   </system.web>
</configuration>
```

By changing the machine key encryption/decryption process to utilize Triple DES, you enable the forms authentication to work across an ASP.NET application that is using both the .NET Framework 1.*x* and 3.5. Also, this example shows the `validationKey` and `decryptionKey` attributes using a specific set of keys. These keys should be the same as those you utilize in your ASP.NET 1.*x* application.

It is important to understand that you are not required to make these changes when you are upgrading an ASP.NET 2.0 application to ASP.NET 3.5 because both are enabled to use AES encryption and are not using Triple DES encryption. If you are mixing an ASP.NET 1.*x* application along with ASP.NET 2.0 and 3.5, then you are going to have to move everything to use Triple DES encryption, as shown in Listing A-1.

Upgrading — ASP.NET Reserved Folders

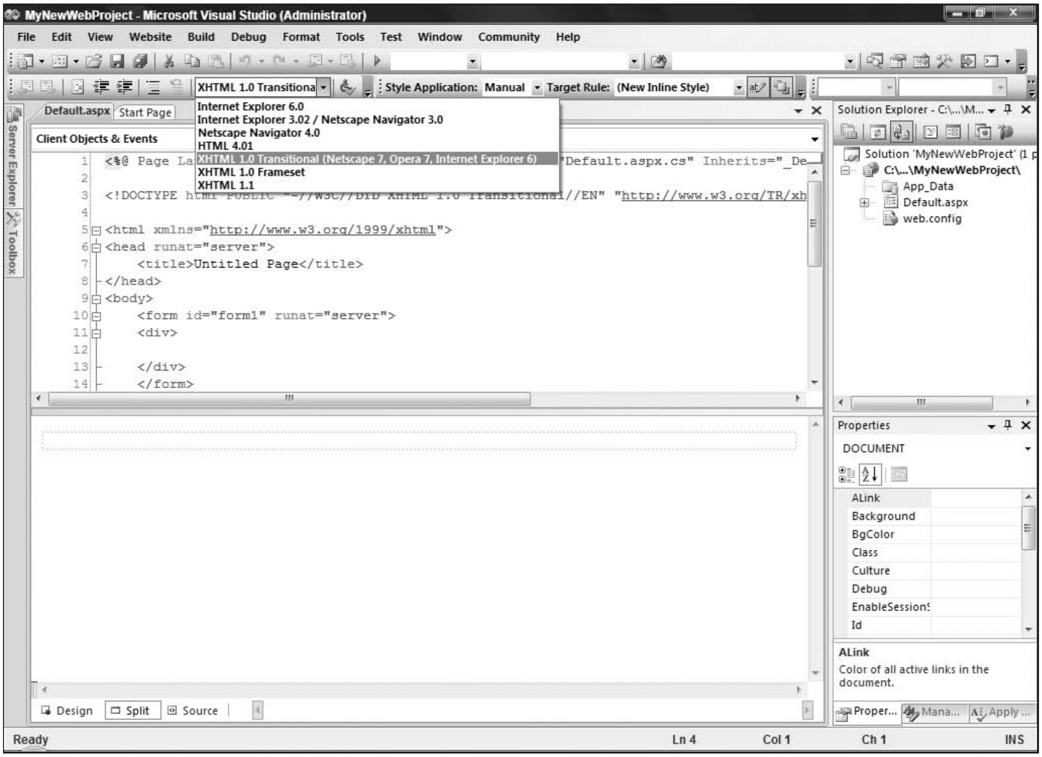
As described in Chapter 1 of this book, ASP.NET 3.5 includes a number of application folders that are specific to the ASP.NET framework. In addition to the `\Bin` folder that was a reserved folder in ASP.NET 1.*x*, the following folders are all reserved in ASP.NET 2.0 and 3.5:

- ☐ `\Bin` : This folder stores the application DLL and any other DLLs used by the application. This folder was present in both ASP.NET 1.0 and 1.1. It is also present in both ASP.NET 2.0 and 3.5.
- ☐ `\App_Code` : This folder is meant to store your classes, `.wsdl` files, and typed datasets. Any items stored in this folder are automatically available to all the pages within your solution.
- ☐ `\App_Data` : This folder holds the data stores utilized by the application. It is a good, central spot to store all the data stores used by your application. The `\App_Data` folder can contain Microsoft SQL Express files (`.mdf files`), Microsoft Access files (`.mdb files`), XML files, and more.
- ☐ `\App_Themes` : Themes are a new way of providing a common look-and-feel to your site across every page. You implement a theme by using a `.skin file`, CSS files, and images used by the server controls of your site. All these elements can make a theme, which is then stored in the `\App_Themes` folder of your solution.
- ☐ `\App_GlobalResources` : This folder enables you to store resource files that can serve as data dictionaries for your applications if these applications require changes in their content (based on things such as changes in culture). You can add Assembly Resource Files (`.resx`) to the `\App_GlobalResources` folder, and they are dynamically compiled and made part of the solution for use by all the .aspx pages in the application.
- ☐ `\App_LocalResources` : Quite similar to the `\App_GlobalResources` folder, the `\App_LocalResources` folder is a pretty simple method to incorporate resources that can be used for a specific page in your application.
- ☐ `\App_WebReferences` : You can use the `\App_WebReferences` folder and have automatic access to the remote Web services referenced from your application.
- ☐ `\App_Browsers` : This folder holds `.browser` files, which are XML files used to identify the browsers making requests to the application and to elucidate the capabilities these browsers have.

The addition of the `App_` prefix to the folder names ensures that you already do not have a folder with a similar name in your ASP.NET 1.*x* applications. If, by chance, you do have a folder with one of the names you plan to use, you should change the name of your previous folder to something else because these ASP.NET 3.5 application folder names are unchangeable.

ASP.NET 3.5 Pages Come as XHTML

ASP.NET 3.5, by default, constructs its pages to be XHTML-compliant. You can see the setting for XHTML 1.0 Transitional in the Visual Studio 2008 IDE. This is illustrated in Figure A-4.


MyNewWebProject - Microsoft Visual Studio (Administrator)
File
Edit
View
Website
Build
Debug
Format
Tools
Test
Window
Community
Help
XHTML 1.0 Transitiona
Internet Explorer 6.0
Internet Explorer 3.02 / Netscape Navigator 3.0
Netscape Navigator 4.0
HTML 4.01
XHTML 1.0 Transitional (Netscape 7, Opera 7, Internet Explorer 6)
XHTML 1.0 Frameset
XHTML 1.1
Server Explorer
Toolbox
Default.aspx
Start Page
Client Objects & Events
Solution Explorer - C:\...\M...
Solution 'MyNewWebProject' (1 p
C:\...\MyNewWebProject\
App_Data
Default.aspx
web.config
Properties
DOCUMENT
ALink
Background
BgColor
Class
Culture
Debug
EnableSession!
Id
ALink
Color of all active links in the document.
Proper...
Mana...
Apply ...
Design
Split
Source
Ready
Ln 4
Col 1
Ch 1
INS

Style Application: Manual
Target Rule: (New Inline Style)

```
<%@ Page La                                       "Default.aspx.cs" Inherits="_De

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xh

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
```

Figure A-4

In this case, you can see that you have a list of options for determining how the ASP.NET application outputs the code for the pages. By default, it is set to `XHTML 1.0 Transitional (Netscape 7, Opera 7, Internet Explorer 6)`. You can also make a change to the `web.config` file so that the output is not XHTML-specific (as illustrated in Listing A-2).

Listing A-2: Reversing the XHTML capabilities of your ASP.NET 3.5 application

```
<configuration>
   <system.web>

      <xhtmlConformance mode="Legacy" />

   </system.web>
</configuration>
```

Setting the `mode` attribute to `Legacy` ensures that XHTML is not used, but instead, ASP.NET 3.5 defaults to what was used in ASP.NET 1.*x*.

It is important to note that using the `Legacy` setting as a value for the `mode` attribute will sometimes cause you problems for your application if you are utilizing AJAX. Some of the symptoms that you might experience is that instead of doing a partial page update (as AJAX does), you will get a full-page postback instead. This is due to the fact that the page is not XHTML compliant. The solution is to set the mode property to `Traditional` or `Strict` and to make your pages XHTML compliant.

If you take this approach, you also have to make some additional changes to any new ASP.NET 3.5 pages that you create in Visual Studio 2008. Creating a new ASP.NET 3.5 page in Visual Studio 2008 produces the results illustrated in Listing A-3.

Listing A-3: A typical ASP.NET 3.5 page

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
</body>
</html>
```

From this, you can see that a `<!DOCTYPE ...>` element is included at the top of the page. This element signifies to some browsers (such as Microsoft's Internet Explorer) that the page is XHTML-compliant. If this is not the case, then you want to remove this element altogether from your ASP.NET 3.5 page. In addition to the `<!DOCTYPE>` element, you also want to change the `<html>` element on the page from:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

to the following:

```
<html>
```

The original also signifies that the page is XHTML-compliant (even if it is not) and must be removed if your pages are not XHTML-compliant.

No Hard-Coded .js Files in ASP.NET 3.5

ASP.NET 1.*x* provides some required JavaScript files as hard-coded `.js` files. For instance, in ASP.NET a JavaScript requirement was necessary for the validation server controls and the smart navigation capabilities to work. If you are utilizing either of these features in your ASP.NET 1.*x* applications, ASP.NET could pick up the installed `.js` files and use them directly.

These `.js` files are found at `C:\WINDOWS\Microsoft.NET\Framework\v1.0.3705\ASP.NETClientFiles`. Looking at this folder, you see three .js files — two of which deal with the smart navigation feature (`SmartNav.js and SmartNavIE5.js`) and one that deals with the validation server controls

(`WebUIValidation.js`). Because they are hard-coded `.js` files, it is possible to open them and change or alter the code in these files to better suit your needs. In some cases, developers have done just that.

If you have altered these JavaScript files in any manner, you must change some code when migrating your ASP.NET application to ASP.NET 2.0 or 3.5. ASP.NET 3.5 dynamically includes .js files from the System.Web.dll instead of hard-coding them on the server. In ASP.NET 3.5, the files are included via a new handler — WebResource.axd.

Converting ASP.NET 1.x Applications in Visual Studio 2008

As previously mentioned, if you have a pre-existing ASP.NET 1.*x* application, you can run the application on the ASP.NET 2.0 runtime by simply making the appropriate changes in IIS to the application pool. Using the IIS manager or the MMC Snap-In, you can select the appropriate framework on which to run your application from the provided drop-down list.

ASP.NET 3.5 applications work with the Visual Studio 2008 IDE. If you still intend to work with ASP.NET 1.0 or 1.1 applications, you should keep Visual Studio .NET 2002 or 2003, respectively, installed on your machine. Installing Visual Studio 2008 gives you a complete, new copy of Visual Studio and does not upgrade the previous Visual Studio .NET 2002 or 2003 IDEs. All copies of Visual Studio can run side by side.

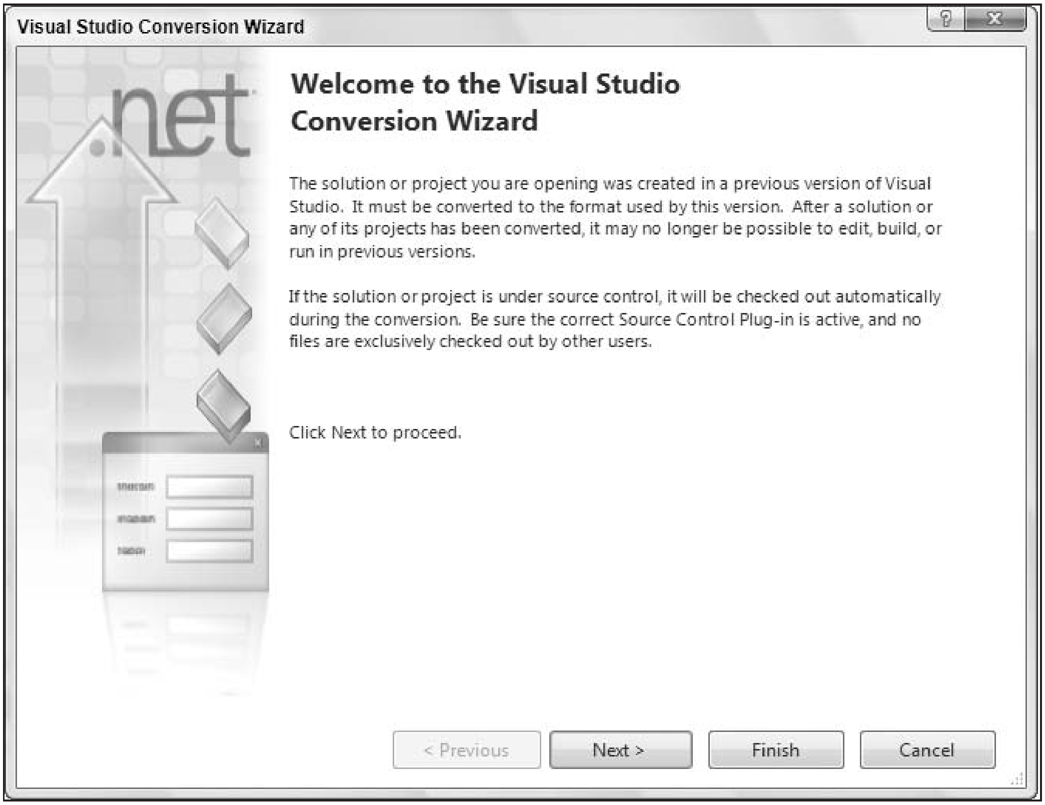
If you want to run ASP.NET 1.*x* applications on the .NET Framework, but you also want to convert the entire ASP.NET project for the application to ASP.NET 3.5, you can use Visual Studio 2008 to help you with the conversion process. After the project is converted in this manner, you can take advantage of the features that ASP.NET 3.5 offers.

To convert your ASP.NET 1.*x* application to an ASP.NET 3.5 application, you simply open up the solution in Visual Studio 2008. This starts the conversion process. It is important to note that Visual Studio 2008 converts the application to an ASP.NET 2.0 application first and then asks if you want to take the extra step to convert the application to an ASP.NET 3.5 application.

When you open the solution in Visual Studio 2008, it warns you that your solution will be upgraded if you continue. It does this by popping up the Visual Studio Conversion Wizard, as presented in Figure A-5.

Notice that the upgrade wizard has been dramatically improved from Visual Studio .NET 2003 to this newer one provided by Visual Studio 2008. To start the conversion process of your ASP.NET 1.*x* applications, click the Next button in the wizard. This example uses the open source Issue Tracker Starter Kit — an ASP.NET 1.1 starter kit found on the ASP.NET Web site at `www.asp.net`.

The first step in the process is deciding whether you want the Visual Studio 2008 Conversion Wizard to create a backup of the ASP.NET 1.1 application before it attempts to convert it to an ASP.NET 2.0 application (remember that it first converts to an ASP.NET 2.0 application before asking you to convert it to an ASP.NET 3.5 application). Definitely, if this is your only copy of the application, you want to make a back-up copy even though this conversion wizard does a good job in the conversion process. The conversion wizard also enables you to specify the location where you want to store the back-up copy. This step is presented in Figure A-6.



Visual Studio Conversion Wizard

Welcome to the Visual Studio Conversion Wizard

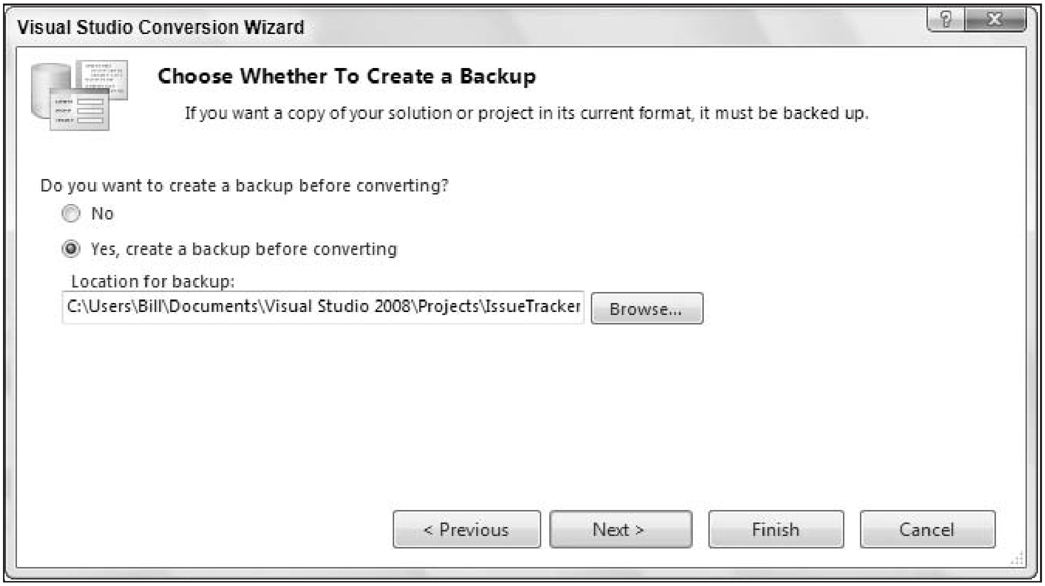
The solution or project you are opening was created in a previous version of Visual Studio. It must be converted to the format used by this version. After a solution or any of its projects has been converted, it may no longer be possible to edit, build, or run in previous versions.

If the solution or project is under source control, it will be checked out automatically during the conversion. Be sure the correct Source Control Plug-in is active, and no files are exclusively checked out by other users.

Click Next to proceed.

< Previous
Next >
Finish
Cancel

Figure A-5



Visual Studio Conversion Wizard

Choose Whether To Create a Backup

If you want a copy of your solution or project in its current format, it must be backed up.

Do you want to create a backup before converting?

○ No

● Yes, create a backup before converting

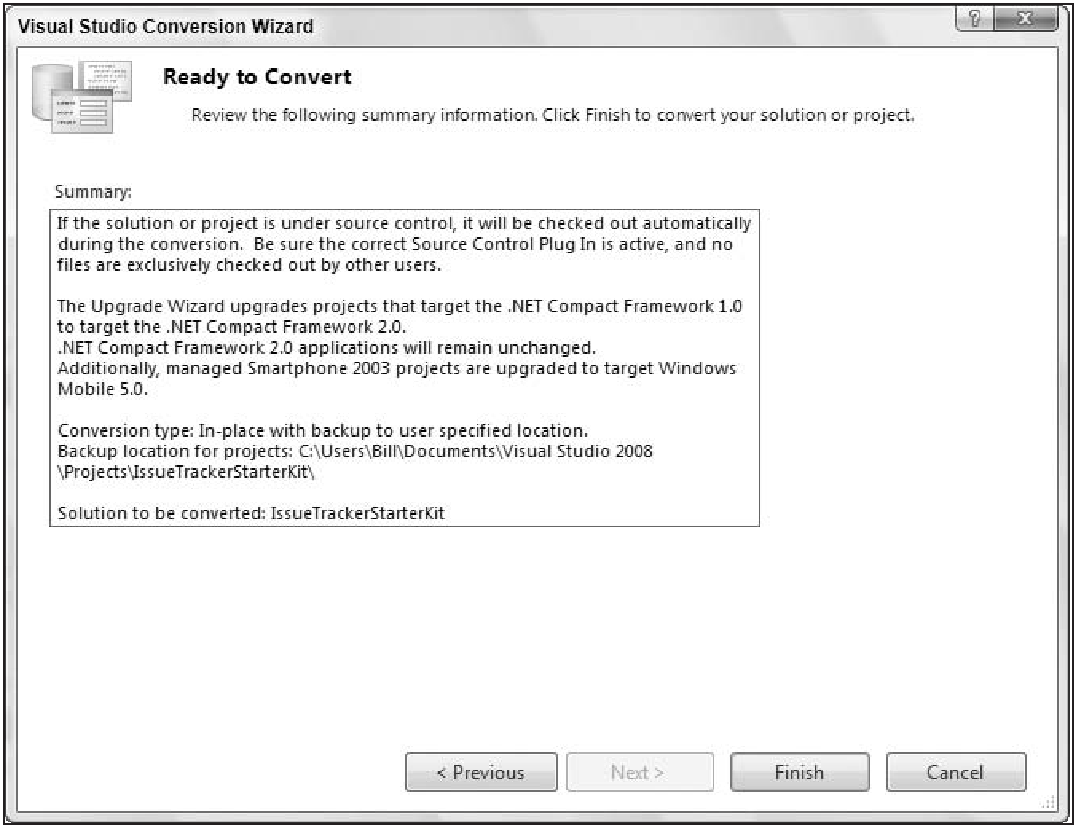
Location for backup:

C:\Users\Bill\Documents\Visual Studio 2008\Projects\IssueTracker

Browse...
< Previous
Next >
Finish
Cancel

Figure A-6

The final step is a warning on how to handle the project if it is controlled by a source control system. If it is, you want to ensure that the project or any of its components are checked out by someone. You also want to ensure that the check-in capabilities are enabled. This warning is shown in Figure A-7.


Visual Studio Conversion Wizard
Ready to Convert
Review the following summary information. Click Finish to convert your solution or project.
Summary:
< Previous
Next >
Finish
Cancel

If the solution or project is under source control, it will be checked out automatically during the conversion. Be sure the correct Source Control Plug In is active, and no files are exclusively checked out by other users.

The Upgrade Wizard upgrades projects that target the .NET Compact Framework 1.0 to target the .NET Compact Framework 2.0.
.NET Compact Framework 2.0 applications will remain unchanged.
Additionally, managed Smartphone 2003 projects are upgraded to target Windows Mobile 5.0.

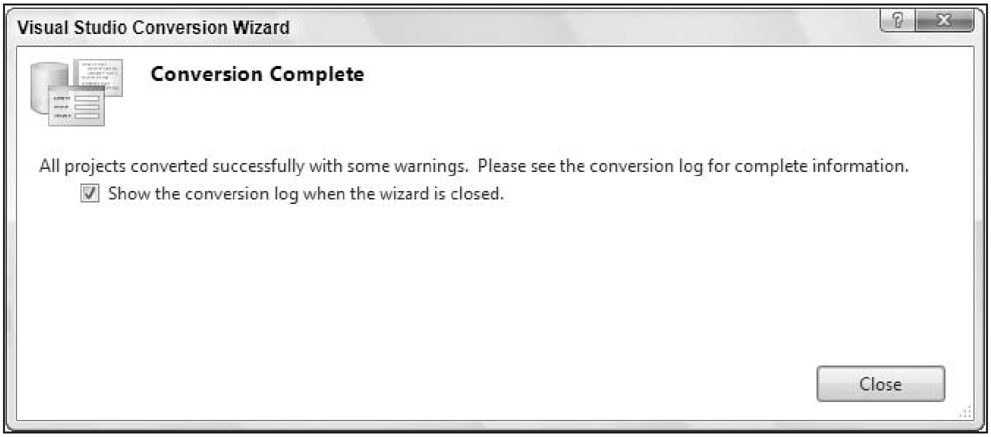
Conversion type: In-place with backup to user specified location.
Backup location for projects: C:\Users\Bill\Documents\Visual Studio 2008 \Projects\IssueTrackerStarterKit\

Solution to be converted: IssueTrackerStarterKit

Figure A-7

When you are ready to convert the project, click the Finish button. The actual conversion process could take some time, so allow a few minutes for it. When the process is complete, you are offered a completion notification that also enables you to see the conversion log that was generated from the conversion process (Figure A-8).

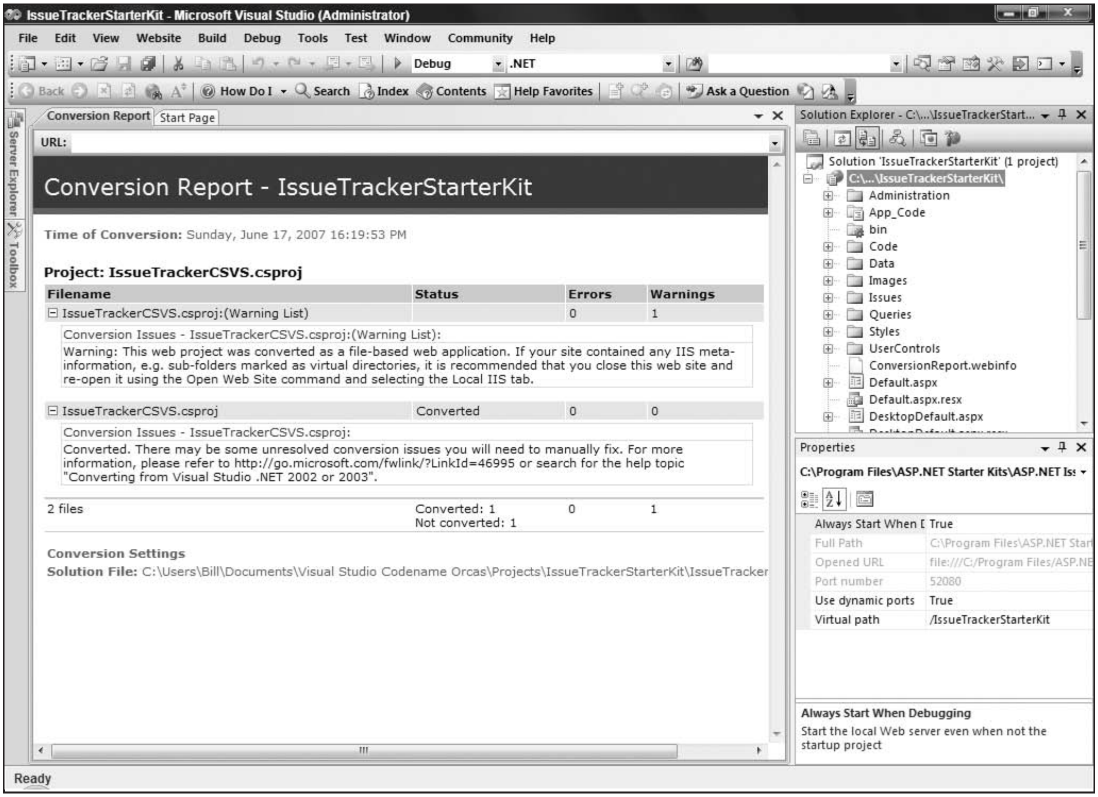
After the project is converted, you are presented with the conversion log, as shown in Figure A-9.


Visual Studio Conversion Wizard
Conversion Complete
Close

All projects converted successfully with some warnings. Please see the conversion log for complete information.

☑ Show the conversion log when the wizard is closed.

Figure A-8


IssueTrackerStarterKit - Microsoft Visual Studio (Administrator)
File
Edit
View
Website
Build
Debug
Tools
Test
Window
Community
Help
Debug
.NET
Back
How Do I
Search
Index
Contents
Help Favorites
Ask a Question
Server Explorer
Toolbox
Conversion Report
Start Page
URL:
Solution Explorer - C:\...\IssueTrackerStart...
Solution 'IssueTrackerStarterKit' (1 project)
C:\...\IssueTrackerStarterKit\
Administration
App_Code
bin
Code
Data
Images
Issues
Queries
Styles
UserControls
ConversionReport.webinfo
Default.aspx
Default.aspx.resx
DesktopDefault.aspx
Properties
Ready

Conversion Report - IssueTrackerStarterKit

Time of Conversion: Sunday, June 17, 2007 16:19:53 PM

Project: IssueTrackerCSVS.csproj

Filename	Status	Errors	Warnings
⊟ IssueTrackerCSVS.csproj:(Warning List)		0	1
Conversion Issues - IssueTrackerCSVS.csproj:(Warning List): Warning: This web project was converted as a file-based web application. If your site contained any IIS meta-information, e.g. sub-folders marked as virtual directories, it is recommended that you close this web site and re-open it using the Open Web Site command and selecting the Local IIS tab.			
⊟ IssueTrackerCSVS.csproj	Converted	0	0
Conversion Issues - IssueTrackerCSVS.csproj: Converted. There may be some unresolved conversion issues you will need to manually fix. For more information, please refer to http://go.microsoft.com/fwlink/?LinkId=46995 or search for the help topic "Converting from Visual Studio .NET 2002 or 2003".			
2 files	Converted: 1 Not converted: 1	0	1

Conversion Settings

Solution File: C:\Users\Bill\Documents\Visual Studio Codename Orcas\Projects\IssueTrackerStarterKit\IssueTracker

C:\Program Files\ASP.NET Starter Kits\ASP.NET Is

Always Start When [	True
Full Path	C:\Program Files\ASP.NET Star
Opened URL	file:///C:/Program Files/ASP.NE
Port number	52080
Use dynamic ports	True
Virtual path	/IssueTrackerStarterKit

Always Start When Debugging

Start the local Web server even when not the startup project

Figure A-9

As you look over the project in the Solution Explorer, notice that some major changes have been made to the project. Some of these changes include the following:

- ❑ All class files are removed from their folders and placed in the new `App_Code` folder. The folder from which the class files were removed is left in place, even if the folder is empty after all the class files are removed.
- ❑ All the Visual Studio .NET 2002/2003 Web project files are deleted because Visual Studio 2008 does not use these any longer.
- ❑ The application's DLL is deleted from the `Bin` folder.
- ❑ All the `.aspx` pages have had their `@Page` directives changed. An example from the `Default.aspx` page shows that the previousc `@Page` directive was constructed as:

```
<%@ Page Language="c#" CodeBehind="Default.aspx.cs" AutoEventWireup="false"
  Inherits="ASPNET.StarterKit.IssueTracker._Default" %>
```

- ❑ After the conversion process, the @Page directive now appears as:

```
<%@ Page Language="c#" Inherits="ASPNET.StarterKit.IssueTracker._Default"
   CodeFile="Default.aspx.cs" %>
```

- ❑ The code-behind classes for the `.aspx` pages are converted to partial classes (presented here in C#). This is what the code behind for the `Default.aspx` page looked like before the conversion:

```
public class _Default : System.Web.UI.Page
{
  // Code removed for clarity
}
```

- ❑ After the conversion process, the page class appears as shown here:

```
public partial class _Default : System.Web.UI.Page
{
  // Code removed for clarity
}
```

For a full list of changes, look for a `ConversionReport.webinfo` file in the root of your solution. The partial text from this example conversion is presented in Listing A-4. This conversion report can get quite large, but pay attention to everything that was done to your project. In the following listing, some of the four pages of this conversion report are shown.

Listing A-4: The ConversionReport.webinfo file

```
This report shows the steps taken to convert your Web application from
ASP.NET 1.1 to ASP.NET 2.0.
There may be some unresolved conversion issues you will need to manually fix.
For more information, please refer to http://go.microsoft.com/fwlink/?LinkId=46995
or search for the help topic "Converting from Visual Studio .NET 2002 or 2003".
```

```
Conversion Started on project file IssueTrackerCSVS.csproj at April 17 2008,
16:48:39.

=========================ERRORS===================

=========================WARNINGS=================
Warning: This web project was converted as a file-based web application. If your
site contained any IIS meta-information, e.g. sub-folders marked as virtual
directories, it is recommended that you close this web site and re-open it
using the Open Web Site command and selecting the Local IIS tab.

=========================COMMENTS=================
Web.Config: Added 'xhtmlConformance' attribute.
Web.Config: added a reference for assembly System.DirectoryServices.
Removed attribute AutoEventWireup from file Default.aspx.
Removed attribute CodeBehind from file Default.aspx.
Removed attribute AutoEventWireup from file DesktopDefault.aspx.
Removed attribute CodeBehind from file DesktopDefault.aspx.
Warning: Access level of 'Page_Load' changed to 'protected' in file
DesktopDefault.aspx.cs (Line 55).
Warning: Access level of 'Login' changed to 'protected' in file
DesktopDefault.aspx.cs (Line 62).
Warning: Access level of 'btnRegister_Click' changed to 'protected' in file
DesktopDefault.aspx.cs (Line 74).
Removed attribute Codebehind from file Global.asax.
Removed attribute AutoEventWireup from file LogOff.aspx.
Removed attribute CodeBehind from file LogOff.aspx.
Warning: Access level of 'Page_Load' changed to 'protected' in file
LogOff.aspx.cs (Line 49).
Removed attribute AutoEventWireup from file NoProjects.aspx.
Removed attribute CodeBehind from file NoProjects.aspx.
Removed attribute AutoEventWireup from file Register.aspx.
Removed attribute CodeBehind from file Register.aspx.
Warning: Access level of 'SaveUser' changed to 'protected' in file
Register.aspx.cs (Line 55).
Removed attribute AutoEventWireup from file administration\projects\addproject.aspx.
Removed attribute CodeBehind from file administration\projects\addproject.aspx.
Warning: Access level of 'Page_Load' changed to 'protected' in file
Removed file Bin\ASPNET.StarterKit.IssueTracker.dll.
Removed file IssueTrackerCSVS.csproj.
Removed file IssueTrackerCSVS.csproj.webinfo.
Project IssueTrackerCSVS.csproj has been converted successfully at April 17 2008,
16:50:12.
```

After the project is converted, you can build and run the application from Visual Studio 2008. The application is now built and run on the ASP.NET 2.0 runtime.

Remember: Do not upgrade production solutions without testing your programs first in a staging environment to ensure that your application is not affected by the changes between versions 1.0/1.1 and 2.0 or 3.5 of the .NET Framework.

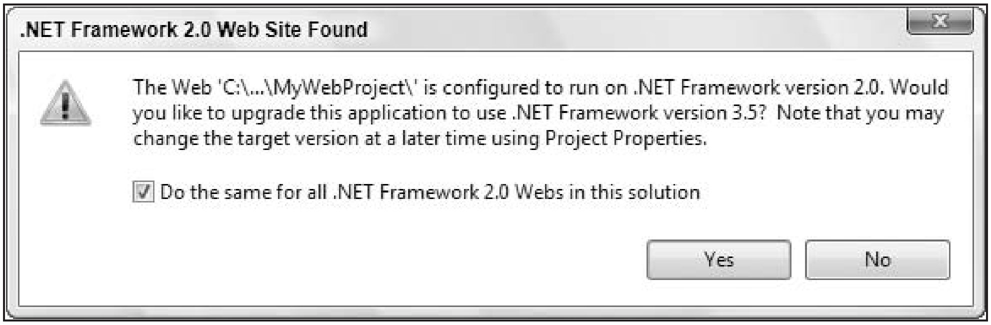
Migrating from ASP.NET 2.0 to 3.5

Visual Studio 2008 is the first version of the IDE that enables you to build applications at more than one framework. For instance, Visual Studio .NET 2002 would only let you build 1.0 applications. If you wanted to build .NET Framework 1.1 applications, then you were required to install and use Visual Studio .NET 2003. At the same time, Visual Studio .NET 2003 would not enable you to build .NET Framework 1.0 applications, meaning that if you were dealing with applications that made use of either framework, then you were required to have both IDEs on your computer.

When you create a new project in Visual Studio 2008, you have the option of targeting the project at any of the following frameworks:

- .NET Framework 2.0
- .NET Framework 3.0
- .NET Framework 3.5

Although you can open your .NET 2.0 applications and work with them directly in Visual Studio 2008, when you first open an ASP.NET 2.0 application in the IDE, you will be prompted to update the application to ASP.NET 3.5. The dialog box that you are presented with is shown in Figure A-10.



.NET Framework 2.0 Web Site Found

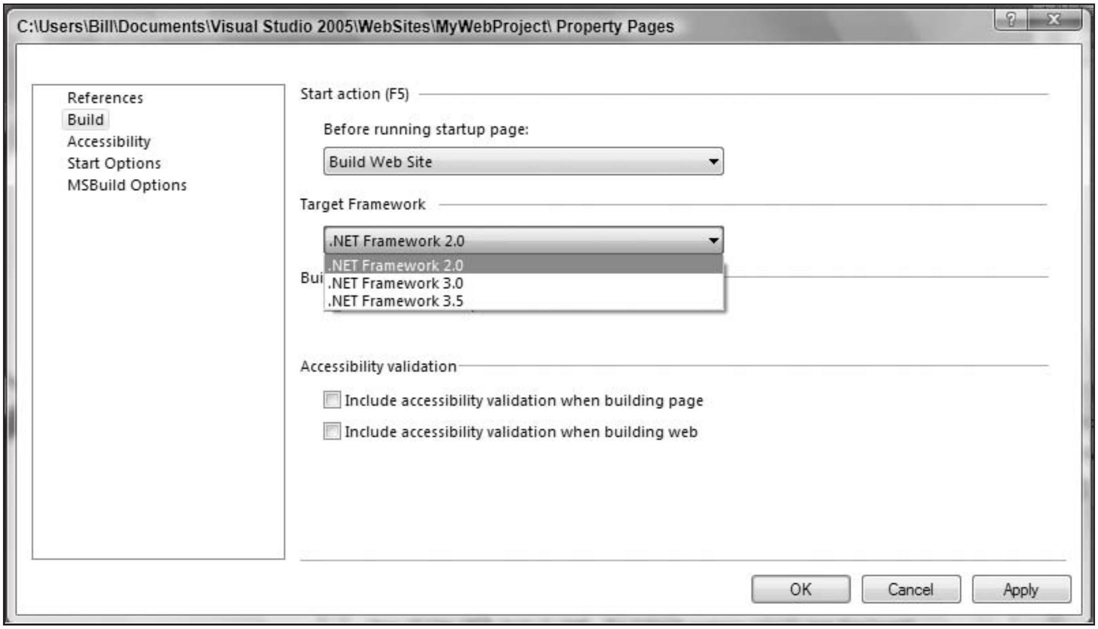
The Web 'C:\...\MyWebProject\' is configured to run on .NET Framework version 2.0. Would you like to upgrade this application to use .NET Framework version 3.5? Note that you may change the target version at a later time using Project Properties.

☑ Do the same for all .NET Framework 2.0 Webs in this solution

Figure A-10

Selecting Yes from this dialog box upgrades your ASP.NET 2.0 application to ASP.NET 3.5. You can also right-click on the project in the Solution Explorer and select Property Pages from the provided menu. This gives you a dialog box that enables you to change the target framework of the application. In this case, you can see the default options on a Microsoft Vista computer (as shown in Figure A-11).

Although you can change the target framework as is illustrated in Figure A-11, you will find that it is better to use Visual Studio 2008, as is shown in Figure A-10, to upgrade your ASP.NET applications. Although ASP.NET 2.0 and ASP.NET 3.5 use the same .NET Framework 2.0 runtime, there are some extra bolted-on additions available to ASP.NET 3.5 applications. The hooks into these extra capabilities are established through changes made by Visual Studio 2008 to the web.config file in the upgrade process. Some of the changes are detailed in the next few listings.



C:\Users\Bill\Documents\Visual Studio 2005\WebSites\MyWebProject\ Property Pages

Start action (F5)

Before running startup page: Build Web Site

Target Framework: .NET Framework 2.0

Accessibility validation

☐ Include accessibility validation when building page

☐ Include accessibility validation when building web

References
Build
Accessibility
Start Options
MSBuild Options
.NET Framework 2.0
.NET Framework 3.0
.NET Framework 3.5
Bui
OK
Cancel
Apply

Figure A-11

The first major change to the web.config file is presented here in Listing A-5.

Listing A-5: Adding the .NET 3.5 language compilers

```
<system.codedom>
   <compilers>
      <compiler language="c#;cs;csharp" extension=".cs"
       type="Microsoft.CSharp.CSharpCodeProvider,System, Version=2.0.0.0,
       Culture=neutral, PublicKeyToken=b77a5c561934e089" compilerOptions="/w:1">
         <providerOption name="CompilerVersion" value="v3.5"/>
      </compiler>
      <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb"
       type="Microsoft.VisualBasic.VBCodeProvider, System, Version=2.0.0.0,
       Culture=neutral, PublicKeyToken=b77a5c561934e089"
       compilerOptions="/optioninfer+">
         <providerOption name="CompilerVersion" value="v3.5"/>
      </compiler>
   </compilers>
</system.codedom>
```

From this bit of the `web.config`, you can see that there are two new compilers provided in this configuration code. Both the C# 3.5 and Visual Basic 3.5 compilers are targeted with ASP.NET 3.5.

The next important change is in the `<compilation>` section of the `web.config`, as shown here in Listing A-6.

Listing A-6: Adding new DLLs to ASP.NET with the 3.5 release

```
<compilation debug="true">
   <assemblies>
      <add assembly="System.Core, Version=2.0.0.0, Culture=neutral,
       PublicKeyToken=B77A5C561934E089"/>
      <add assembly="System.Web.Extensions, Version=2.0.0.0, Culture=neutral,
       PublicKeyToken=31BF3856AD364E35"/>
   </assemblies>
</compilation>
```

In ASP.NET 3.5, the `System.Core` and `System.Web.Extensions` DLLs are added and made available to this new version of the framework.

In addition to these two major additions to the `web.config`, you will find large sections of other changes that mainly deal with the new AJAX capabilities that ASP.NET 3.5 provides. If you are not using ASP.NET AJAX in your applications, you can then delete these sections from the configuration file.

B

ASP.NET Ultimate Tools

I've always believed that I'm only as good as my tools. I've spent years combing the Internet for excellent tools to help me be a more effective developer. There are thousands of tools out there to be sure, many overlapping in functionality with others. Some tools do one thing incredibly well and others aim to be a Swiss Army Knife with dozens of small conveniences packed into their tiny toolbars. Here is a short, exclusive list of some of the ASP.NET tools that I keep turning back to. These are tools that I find myself using consistently while developing ASP.NET-based Web sites. I recommend that you give them a try if they sound useful. Many are free; some are not. In my opinion, each is worth at least a trial on your part, and many are worth your hard earned money as they'll save you precious time.

These tools can be easily searched for in your favorite search engine and found in the first page. For those that are harder to find, I've included URLs. I also encourage you to check out my annually updated Ultimate Tools List at `www.hanselman.com/tools` and you might also enjoy my weekly podcast at `www.hanselminutes.com` as we often discover and share new tools for the developer enthusiast.

Enjoy!

–Scott Hanselman

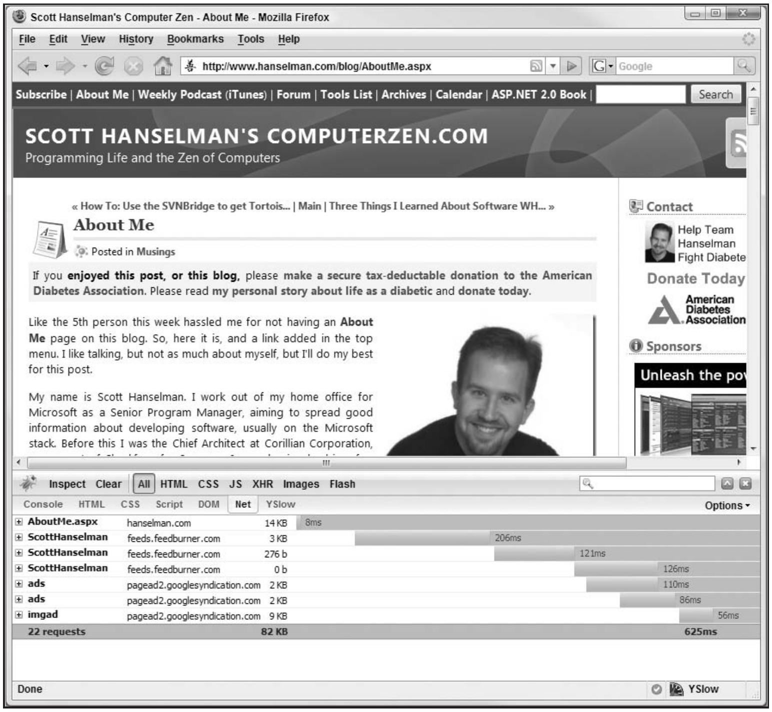
Debugging Made Easier

> ''There has never been an unexpectedly short debugging period in the history of computers.''
> — Steven Levy

Firebug

There are so many great things about this application one could write a book about it. Firebug is actually a Firefox plug-in, so you'll need to download and install Firefox to use it.

The screenshot below shows Firebug analyzing all the network traffic required to download my page. This shows a very detailed graph of when each asset is downloaded and how long it took from first byte to last byte as seen in Figure B-1.


Scott Hanselman's Computer Zen - About Me - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.hanselman.com/blog/AboutMe.aspx
Google
Subscribe | About Me | Weekly Podcast (iTunes) | Forum | Tools List | Archives | Calendar | ASP.NET 2.0 Book |
Search
SCOTT HANSELMAN'S COMPUTERZEN.COM
Programming Life and the Zen of Computers
« How To: Use the SVNBridge to get Tortois... | Main | Three Things I Learned About Software WH... »
About Me
Posted in Musings
Contact
Help Team Hanselman Fight Diabete
Donate Today
American Diabetes Association
Sponsors
Unleash the po
Inspect Clear All HTML CSS JS XHR Images Flash
Console HTML CSS Script DOM Net YSlow
Options
Done
YSlow

If you **enjoyed this post, or this blog,** please **make a secure tax-deductable donation to the American Diabetes Association**. Please read **my personal story about life as a diabetic** and **donate today**.

Like the 5th person this week hassled me for not having an **About Me** page on this blog. So, here it is, and a link added in the top menu. I like talking, but not as much about myself, but I'll do my best for this post.

My name is Scott Hanselman. I work out of my home office for Microsoft as a Senior Program Manager, aiming to spread good information about developing software, usually on the Microsoft stack. Before this I was the Chief Architect at Corillian Corporation,

```
AboutMe.aspx     hanselman.com                  14 KB   8ms
ScottHanselman   feeds.feedburner.com            3 KB   206ms
ScottHanselman   feeds.feedburner.com          276 b    121ms
ScottHanselman   feeds.feedburner.com            0 b    126ms
ads              pagead2.googlesyndication.com   2 KB   110ms
ads              pagead2.googlesyndication.com   2 KB   86ms
imgad            pagead2.googlesyndication.com   9 KB   56ms
22 requests                                     82 KB   625ms
```

Figure B-1

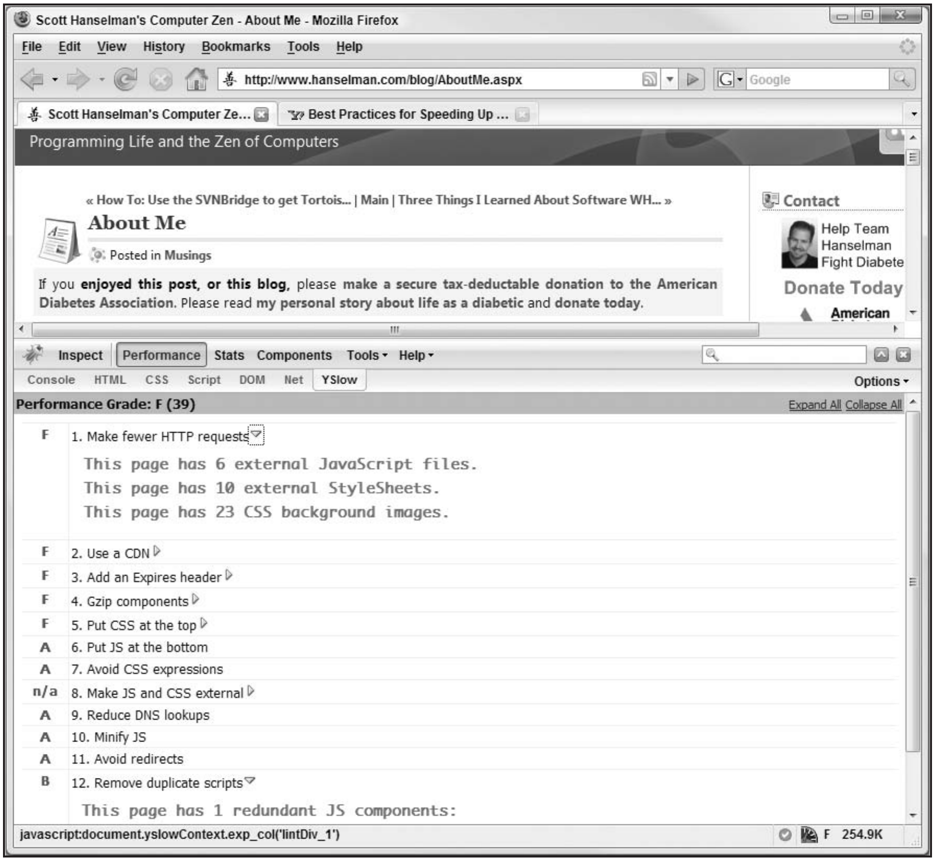
It has a wealth of interesting features that allow you to inspect HTML and deeply analyze your CSS including visualization of some more complicated CSS techniques such as offsets, margins, borders, and padding. Firebug also includes a powerful JavaScript debugger that will enable you to debug JavaScript within Firefox. Even more interesting is its JavaScript profiler and a very detailed error handler that helps you chase down even the most obscure bugs.

Finally, Firebug includes an interactive console feature like the Visual Studio Immediate window that lets you execute JavaScript on-the-fly, as well as console debugging that enables classic "got here" debugging. Firebug is indispensable for the Web developer and it's highly recommended.

There is also Firebug Lite in the form of a JavaScript file. You can add it to the pages in which you want a console debugger to work in Internet Explorer, Opera, or Safari. This file will enable you to do "got here" debugging using the Firebug JavaScript `console.log` method.

YSlow

YSlow is an add-on to an add-on. Brought to you by Yahoo!, YSlow extends Firebug and analyzes your Web pages using Yahoo's 13 rules for fast Web sites. In Figure B-2, you can see Yahoo's YSlow analyzing my blog.


Scott Hanselman's Computer Zen - About Me - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.hanselman.com/blog/AboutMe.aspx
Google
Scott Hanselman's Computer Ze...
Best Practices for Speeding Up ...
Programming Life and the Zen of Computers
« How To: Use the SVNBridge to get Tortois... | Main | Three Things I Learned About Software WH... »
About Me
Posted in Musings
Contact
Help Team Hanselman Fight Diabete
Donate Today
American
Inspect Performance Stats Components Tools Help
Console HTML CSS Script DOM Net YSlow
Options
Expand All Collapse All
javascript:document.yslowContext.exp_col('lintDiv_1')
F 254.9K

If you enjoyed this post, or this blog, please make a secure tax-deductible donation to the American Diabetes Association. Please read my personal story about life as a diabetic and donate today.

Performance Grade: F (39)

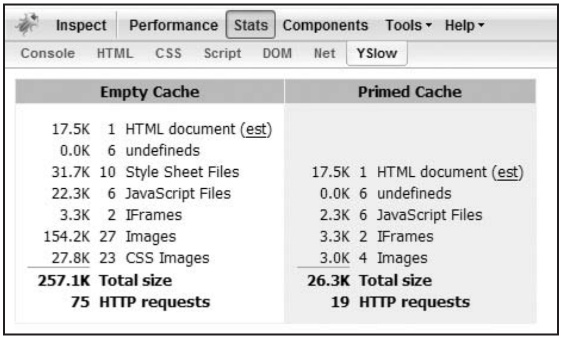
F	1. Make fewer HTTP requests
	This page has 6 external JavaScript files. This page has 10 external StyleSheets. This page has 23 CSS background images.
F	2. Use a CDN
F	3. Add an Expires header
F	4. Gzip components
F	5. Put CSS at the top
A	6. Put JS at the bottom
A	7. Avoid CSS expressions
n/a	8. Make JS and CSS external
A	9. Reduce DNS lookups
A	10. Minify JS
A	11. Avoid redirects
B	12. Remove duplicate scripts
	This page has 1 redundant JS components:

Figure B-2

In some instances, I do well, but in others I receive a failing grade. For example, rule number one says to make fewer HTTP requests. My site has too many external assets. Each one of these requires an HTTP request, so I suspect I could speed up my site considerably with some refactoring.

Not every rule will apply to you exactly, but Yahoo! knows what they're doing and it's worth your time to use this tool and consider your grades in each category. At the very least, you'll gain insight into how your application behaves. For example, Figure B-3 shows how many HTTP requests and bytes are transmitted with an empty cache versus a primed one.

YSlow is free and is an excellent resource to help you get a clear understanding about how hard the client's browser must work in order to view your Web site.


Inspect
Performance
Stats
Components
Tools
Help
Console
HTML
CSS
Script
DOM
Net
YSlow

Empty Cache			Primed Cache		
17.5K	1	HTML document (est)			
0.0K	6	undefineds			
31.7K	10	Style Sheet Files	17.5K	1	HTML document (est)
22.3K	6	JavaScript Files	0.0K	6	undefineds
3.3K	2	IFrames	2.3K	6	JavaScript Files
154.2K	27	Images	3.3K	2	IFrames
27.8K	23	CSS Images	3.0K	4	Images
257.1K	**Total size**		**26.3K**	**Total size**	
75	**HTTP requests**		**19**	**HTTP requests**	

Figure B-3

IE WebDeveloper Toolbar and Firefox WebDeveloper

Both of these toolbars are free and absolutely essential for Web development. The IE Web Developer Toolbar is from Microsoft and extends Internet Explorer with a docked ‘‘Explorer Bar’’ offering features such as DOM inspection and element outlining. You can edit the CSS and see what styles are applied to specific elements as seen in Figure B-4.

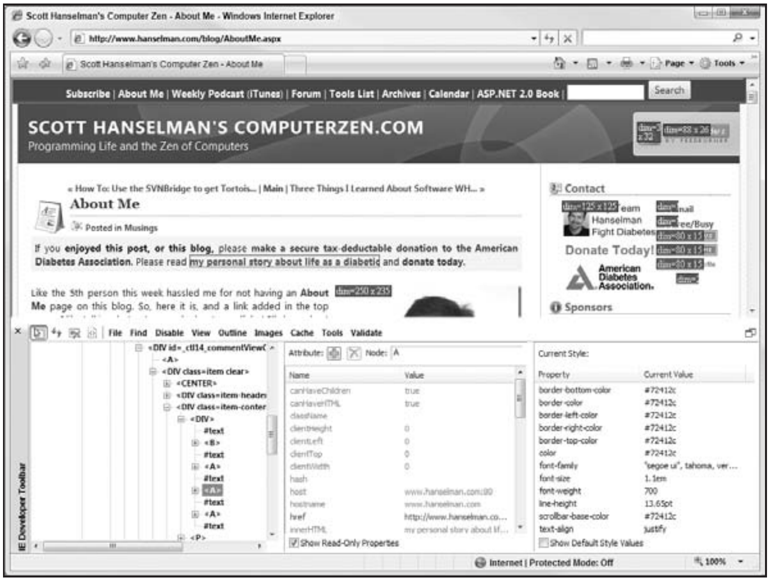
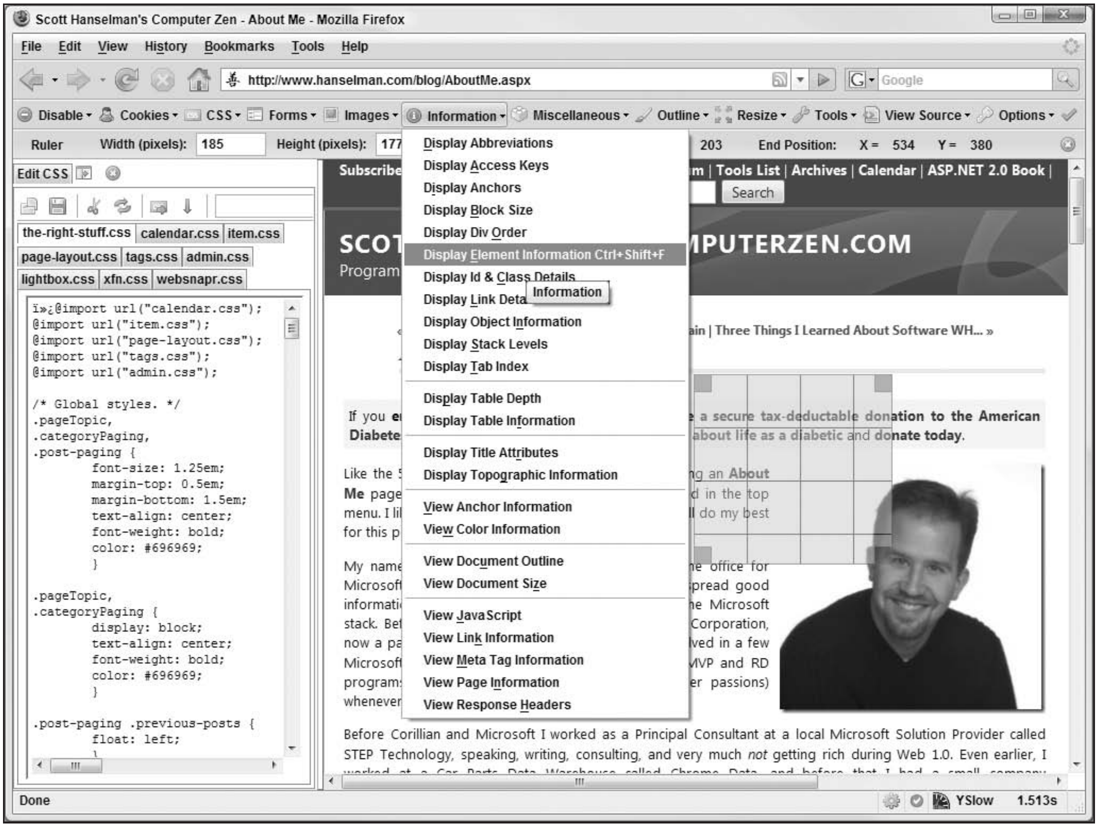

Scott Hanselman's Computer Zen - About Me - Windows Internet Explorer
http://www.hanselman.com/blog/AboutMe.aspx
Scott Hanselman's Computer Zen - About Me
Page
Tools
Subscribe | About Me | Weekly Podcast (iTunes) | Forum | Tools List | Archives | Calendar | ASP.NET 2.0 Book |
Search
SCOTT HANSELMAN'S COMPUTERZEN.COM
Programming Life and the Zen of Computers
About Me
Contact
Donate Today!
American Diabetes Association.
Sponsors
File
Find
Disable
View
Outline
Images
Cache
Tools
Validate
Attribute:
Node:
Name
Value
Current Style:
Property
Current Value
Show Read-Only Properties
Show Default Style Values
IE Developer Toolbar
Internet | Protected Mode: Off
100%

Figure B-4

Firefox has a similar but even more powerful Web Developer Toolbar created by Chris Pederick. It takes a slightly different direction for its user interface by including a number of menus, each literally chock full of menu options. You can disable cookies, CSS, images, inspect elements, form inputs, and outline tables, as shown in Figure B-5.


Scott Hanselman's Computer Zen - About Me - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.hanselman.com/blog/AboutMe.aspx
Google
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options
Ruler Width (pixels): 185 Height (pixels): 177 203 End Position: X = 534 Y = 380
Edit CSS
the-right-stuff.css calendar.css item.css
page-layout.css tags.css admin.css
lightbox.css xfn.css websnapr.css
Display Abbreviations
Display Access Keys
Display Anchors
Display Block Size
Display Div Order
Display Element Information Ctrl+Shift+F
Display Id & Class Details
Display Link Deta
Information
Display Object Information
Display Stack Levels
Display Tab Index
Display Table Depth
Display Table Information
Display Title Attributes
Display Topographic Information
View Anchor Information
View Color Information
View Document Outline
View Document Size
View JavaScript
View Link Information
View Meta Tag Information
View Page Information
View Response Headers
Subscribe
m | Tools List | Archives | Calendar | ASP.NET 2.0 Book |
Search
SCOT
PUTERZEN.COM
Program
ain | Three Things I Learned About Software WH... »
Done
YSlow 1.513s

```
ï»¿@import url("calendar.css");
@import url("item.css");
@import url("page-layout.css");
@import url("tags.css");
@import url("admin.css");

/* Global styles. */
.pageTopic,
.categoryPaging,
.post-paging {
        font-size: 1.25em;
        margin-top: 0.5em;
        margin-bottom: 1.5em;
        text-align: center;
        font-weight: bold;
        color: #696969;
        }

.pageTopic,
.categoryPaging {
        display: block;
        text-align: center;
        font-weight: bold;
        color: #696969;
        }

.post-paging .previous-posts {
        float: left;
```

Figure B-5

ASP.NET developers today need their sites to look great in both browsers. These two toolbars put a host of usefulness at your fingertips. Highly recommended.

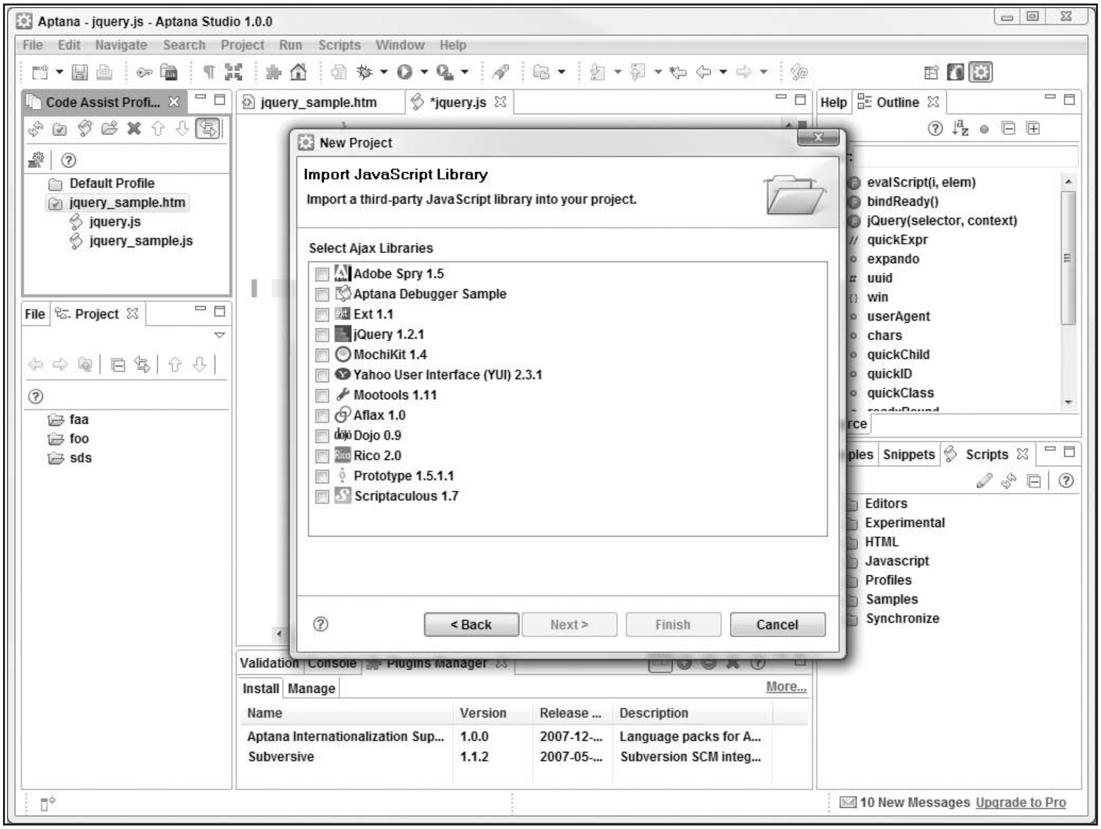
Aptana Studio — Javascript IDE

Aptana offers a pay-for Professional Edition and a Free Community Edition. It's a Web development environment based on the Eclipse codebase. It's not optimized for ASP.NET, but it includes some amazing JavaScript-specific support that makes it worth looking at for AJAX or JavaScript heavy sites.

One of Aptana's most compelling features is its understanding of browser compatibility of JavaScript properties and functions. Aptana's IntelliSense includes icons for the various browsers and dims them out appropriately for unsupported features, for not only JavaScript, but also HTML and CSS, as well.

The Professional Edition also includes a JSON (JavaScript Object Notation) editor with syntax highlighting for JSON datasets. This is a great feature if you're passing a large and deep dataset back and forth via AJAX.

Aptana also includes all the major open source AJAX libraries such as Prototype and Scriptaculous, as shown in Figure B-6. Adding support for them simply requires checking a check box. It also integrates with Firebug. The Pro edition integrates with Internet Explorer.


Aptana - jquery.js - Aptana Studio 1.0.0
File
Edit
Navigate
Search
Project
Run
Scripts
Window
Help
Code Assist Profi...
Default Profile
jquery_sample.htm
jquery.js
jquery_sample.js
File
Project
faa
foo
sds
jquery_sample.htm
*jquery.js
Help
Outline
evalScript(i, elem)
bindReady()
jQuery(selector, context)
quickExpr
expando
uuid
win
userAgent
chars
quickChild
quickID
quickClass
rce
ples
Snippets
Scripts
Editors
Experimental
HTML
Javascript
Profiles
Samples
Synchronize
Validation
Console
Plugins Manager
Install
Manage
More...
10 New Messages
Upgrade to Pro

New Project

Import JavaScript Library

Import a third-party JavaScript library into your project.

Select Ajax Libraries

- ☐ Adobe Spry 1.5
- ☐ Aptana Debugger Sample
- ☐ Ext 1.1
- ☐ jQuery 1.2.1
- ☐ MochiKit 1.4
- ☐ Yahoo User Interface (YUI) 2.3.1
- ☐ Mootools 1.11
- ☐ Aflax 1.0
- ☐ Dojo 0.9
- ☐ Rico 2.0
- ☐ Prototype 1.5.1.1
- ☐ Scriptaculous 1.7

< Back
Next >
Finish
Cancel

Name	Version	Release ...	Description
Aptana Internationalization Sup...	1.0.0	2007-12-...	Language packs for A...
Subversive	1.1.2	2007-05-...	Subversion SCM integ...

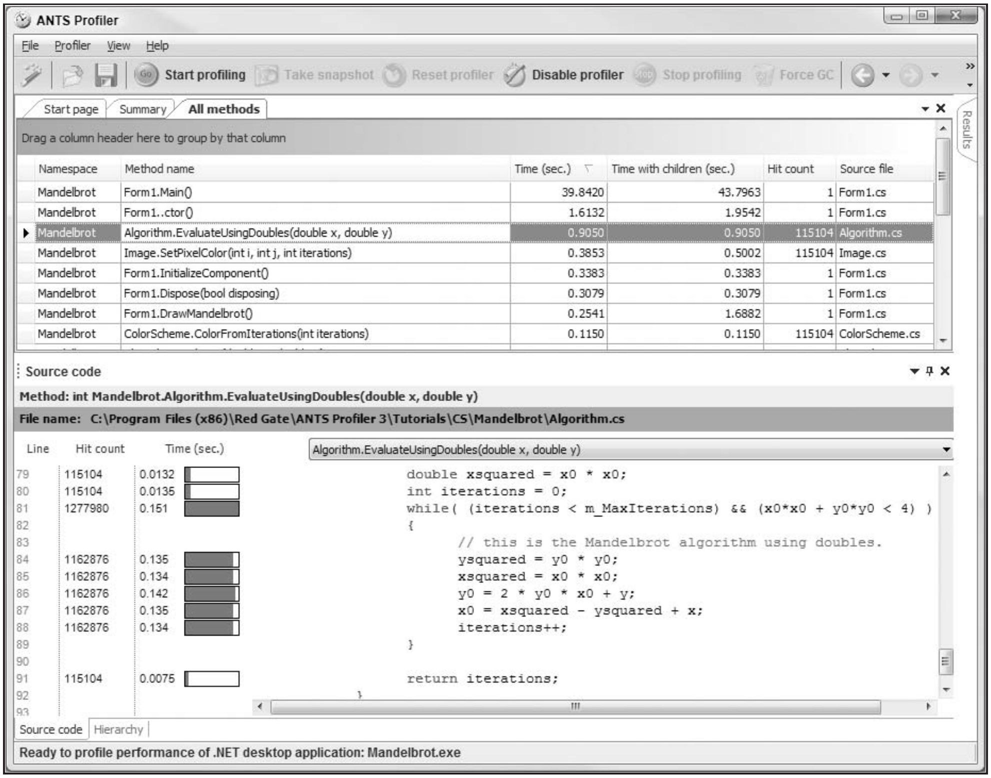
Figure B-6

Profilers: dotTrace or ANTS

If you're not measuring your code with a good profiler you really don't realize what you're missing out on. Only a profiler can give you extensive metrics and a clear understanding of what your code is doing.

Visual Studio Team System 2008 includes a Profiler in the top-level Analyze menu. If you're not running VSTS, there are excellent third party profilers such as Jetbrain's dotTrace and Red Gate Software's ANTS that are worth your 10-day trial.

.NET Profilers instruments a runtime session of your code and measure how many times each line is hit and how much time is spent executing that line, as shown in Figure B-7. They create a hierarchical series of reports that allow you to analyze time spent not only within a method, but within child methods executed through the stack. Reports can be saved and multiple versions can be analyzed as you improve your applications, revision after revision.



Namespace	Method name	Time (sec.)	Time with children (sec.)	Hit count	Source file
Mandelbrot	Form1.Main()	39.8420	43.7963	1	Form1.cs
Mandelbrot	Form1..ctor()	1.6132	1.9542	1	Form1.cs
Mandelbrot	Algorithm.EvaluateUsingDoubles(double x, double y)	0.9050	0.9050	115104	Algorithm.cs
Mandelbrot	Image.SetPixelColor(int i, int j, int iterations)	0.3853	0.5002	115104	Image.cs
Mandelbrot	Form1.InitializeComponent()	0.3383	0.3383	1	Form1.cs
Mandelbrot	Form1.Dispose(bool disposing)	0.3079	0.3079	1	Form1.cs
Mandelbrot	Form1.DrawMandelbrot()	0.2541	1.6882	1	Form1.cs
Mandelbrot	ColorScheme.ColorFromIterations(int iterations)	0.1150	0.1150	115104	ColorScheme.cs

Method: int Mandelbrot.Algorithm.EvaluateUsingDoubles(double x, double y)

File name: C:\Program Files (x86)\Red Gate\ANTS Profiler 3\Tutorials\CS\Mandelbrot\Algorithm.cs

Algorithm.EvaluateUsingDoubles(double x, double y)

```
Line  Hit count  Time (sec.)
79    115104     0.0132          double xsquared = x0 * x0;
80    115104     0.0135          int iterations = 0;
81    1277980    0.151           while( (iterations < m_MaxIterations) && (x0*x0 + y0*y0 < 4) )
82                               {
83                                     // this is the Mandelbrot algorithm using doubles.
84    1162876    0.135                 ysquared = y0 * y0;
85    1162876    0.134                 xsquared = x0 * x0;
86    1162876    0.142                 y0 = 2 * y0 * x0 + y;
87    1162876    0.135                 x0 = xsquared - ysquared + x;
88    1162876    0.134                 iterations++;
89                               }
90
91    115104     0.0075          return iterations;
92                         }
93
```

Ready to profile performance of .NET desktop application: Mandelbrot.exe

Figure B-7

If you haven't already done so, consider adding profiling of your ASP.NET application to your software development lifecycle. You'd be surprised to learn how few developers formally analyze and profile their applications. Set aside some time to profile an application that you've never looked at before and you'll be surprised how much faster it can be made using analysis from a tool such as ANTS or dotTrace.

References

''He who lends a book is an idiot. He who returns the book is more of an idiot.'' — Anonymous, Arabic Proverb

PositionIsEverything.net, QuirksMode.org, and HTMLDog.com

When you're creating Web sites that need to look nice on all browsers, you're bound to bump into bugs, ''features,'' and general differences in the popular browsers. Web pages are composed of a large combination of standards (HTML, CSS, JS). These standards are not only open to interpretation, but their implementations can differ in subtle ways, especially when they interact.

Reference Web sites, such as PositionIsEverything and QuirksMode, collect hundreds of these hacks and workarounds. Then they catalog them for your benefit. Many of these features aren't designed, but rather discovered or stumbled upon.

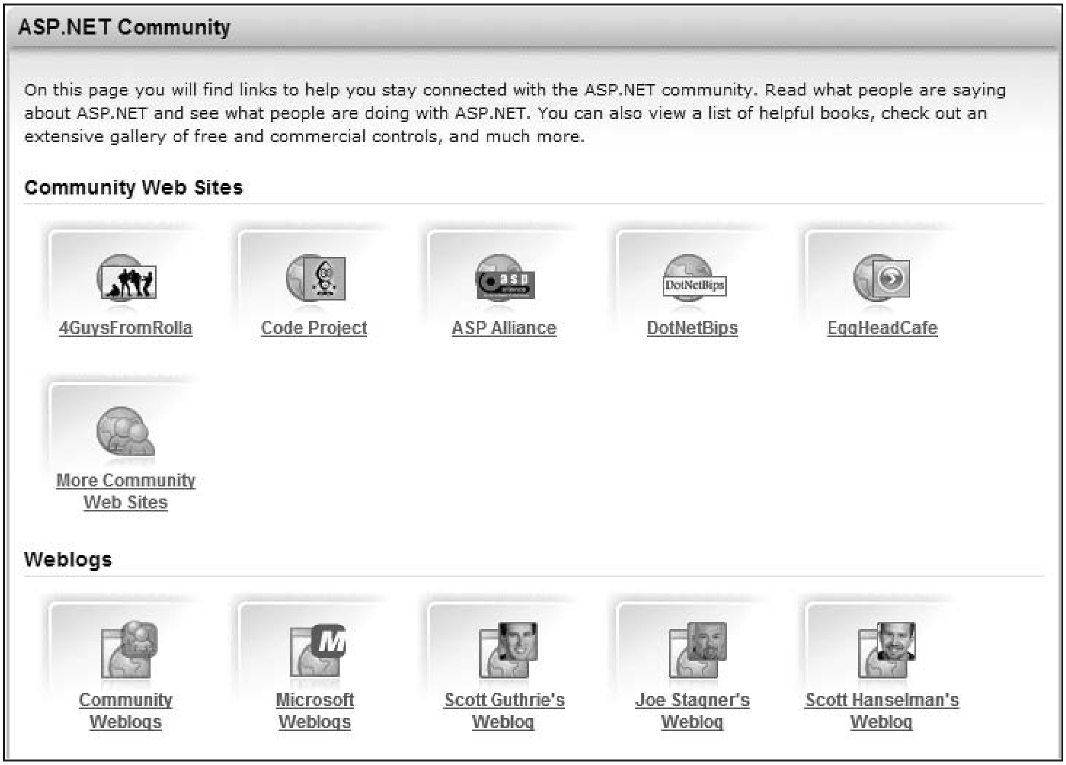
HTMLDog is a fantastic Web designer's resource for HTML and CSS. It's full of tutorials, articles, and a large reference section specific to XHTML. QuirksMode includes many resources for learning JavaScript and CSS and includes many test and demo pages demonstrating the quirks. PositionIsEverything is hosted by John and Holly Bergevin and showcases some of the most obscure bugs and browser oddities with demo examples for everything.

Visibone

Visibone is known for its amazing reference cards and charts that showcase Color, Fonts, HTML, JavaScript, and CSS. Visibone reference cards and booklets are available online and are very reasonably priced. The best value is the Browser Book available at `www.visibone.com/products/browserbook.html`. I recommend the laminated version. Be sure to put your name on it because your co-workers will make it disappear.

www.asp.net

I work for Microsoft on the team that runs `www.asp.net`. The site is a huge resource for learning about ASP.NET and the various technologies around it. Figure B-8 shown part of the Community page for the site, where you'll a link to my Weblog, among others, and links to other community resources. The `www.asp.net/learn/` section includes dozens and dozens of videos about general ASP.NET and how to use it.



ASP.NET Community

On this page you will find links to help you stay connected with the ASP.NET community. Read what people are saying about ASP.NET and see what people are doing with ASP.NET. You can also view a list of helpful books, check out an extensive gallery of free and commercial controls, and much more.

Community Web Sites

4GuysFromRolla
Code Project
ASP Alliance
DotNetBips
EggHeadCafe
More Community Web Sites

Weblogs

Community Weblogs
Microsoft Weblogs
Scott Guthrie's Weblog
Joe Stagner's Weblog
Scott Hanselman's Weblog

Figure B-8

Tidying Up Your Code

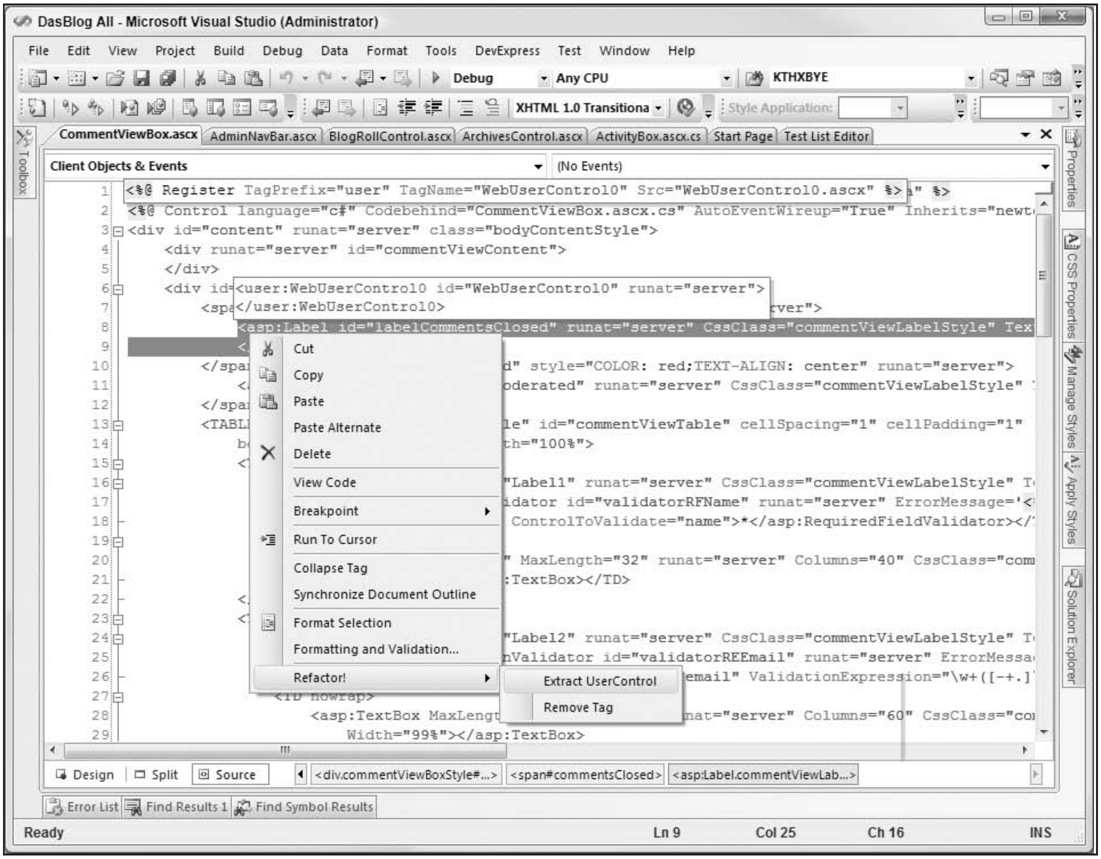
''After every war someone has to tidy up.'' — Wislawa Szymborska

Refactor! for ASP.NET from Devexpress

Refactoring support in Visual Studio 2008 continues to get better. The third party utilities continue to push the envelope adding value to the IDE. Refactor! for ASP.NET adds refactoring to the ASP.NET source view.

For example, in Figure B-9 while hovering over the Refactor! context menu and selecting the ''Extract UserControl'' refactoring, a preview of the changes that *would* be made appear within the source view. A new UserControl would be created in a new file `WebUserControl0.ascx`. The currently selected label control would turn into a `WebUserControl0` control. You can then choose a new name for the UserControl immediately after the refactoring.

The most amazing thing about Refactor! for ASP.NET is that it's a free download from `www.devexpress.com/Products/NET/IDETools/RefactorASP/`. It includes 28 refactorings that make it easier to simplify your code and your ASP.NET markup.



DasBlog All - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Data Format Tools DevExpress Test Window Help
Debug
Any CPU
KTHXBYE
XHTML 1.0 Transitiona
Style Application:
Toolbox
CommentViewBox.ascx
AdminNavBar.ascx
BlogRollControl.ascx
ArchivesControl.ascx
ActivityBox.ascx.cs
Start Page
Test List Editor
Client Objects & Events
(No Events)
Cut
Copy
Paste
Paste Alternate
Delete
View Code
Breakpoint
Run To Cursor
Collapse Tag
Synchronize Document Outline
Format Selection
Formatting and Validation...
Refactor!
Extract UserControl
Remove Tag
Design
Split
Source
<div.commentViewBoxStyle#...>
<span#commentsClosed>
<asp:Label.commentViewLab...>
Error List
Find Results 1
Find Symbol Results
Ready
Ln 9
Col 25
Ch 16
INS
Properties
CSS Properties
Manage Styles
Apply Styles
Solution Explorer

```
<%@ Register TagPrefix="user" TagName="WebUserControl0" Src="WebUserControl0.ascx" %>a" %>
<%@ Control language="c#" Codebehind="CommentViewBox.ascx.cs" AutoEventWireup="True" Inherits="newt
<div id="content" runat="server" class="bodyContentStyle">
    <div runat="server" id="commentViewContent">
    </div>
    <div id<user:WebUserControl0 id="WebUserControl0" runat="server">
        <spa</user:WebUserControl0>                                    rver">
            <asp:Label id="labelCommentsClosed" runat="server" CssClass="commentViewLabelStyle" Tex
            <
        </spa                     d" style="COLOR: red;TEXT-ALIGN: center" runat="server">
            <                     oderated" runat="server" CssClass="commentViewLabelStyle"
        </spa
        <TABL                     le" id="commentViewTable" cellSpacing="1" cellPadding="1"
            b                     th="100%">
            <

                                  "Label1" runat="server" CssClass="commentViewLabelStyle" T
                                  idator id="validatorRFName" runat="server" ErrorMessage='<
                                   ControlToValidate="name">*</asp:RequiredFieldValidator></

                                  " MaxLength="32" runat="server" Columns="40" CssClass="com
                                  :TextBox></TD>
            <
            <
                                  "Label2" runat="server" CssClass="commentViewLabelStyle" T
                                  nValidator id="validatorREEmail" runat="server" ErrorMessa
                                                  email" ValidationExpression="\w+([-+.]
                <TD nowrap>
                    <asp:TextBox MaxLengt                   nat="server" Columns="60" CssClass="co
                        Width="99%"></asp:TextBox>
```

Figure B-9

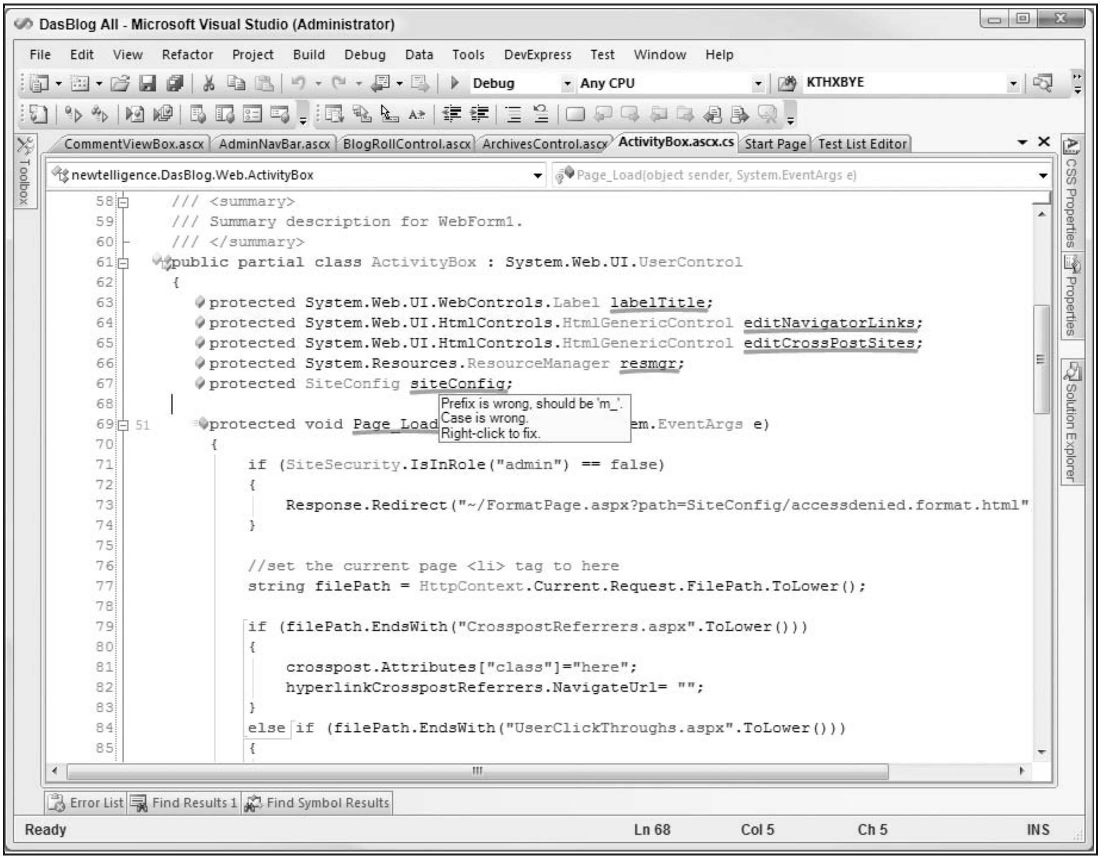
Code Style Enforcer

Code Style Enforcer from Joel Fjordén does just that. It's a DXCore Plugin that enforces code style rules that you configure. DXCore is the free engine from DevExpress that Refactor! uses to extend Visual Studio.

Every team has a coding standard that they'd like programmers to follow, but it's not only hard to keep track of all the rules, it's tedious. Are methods supposed to be CamelCased or pascalCased? Are we putting ''m_'' in front of member fields?

Code Style Enforcer is a lot like Microsoft Word's spelling and grammar checker except for code. As shown in Figure B-10, identifiers that do not meet the code style guidelines are underlined in red. You can right-click on each error and Code Style Enforcer will use DxCore to refactor and fix each violation.

Style guidelines are configurable and the default uses Juval Lowy's excellent C# Code Style Guidelines available from `www.idesign.net`. The latest version will also generate code rule violation reports for a solution using XML and XSLT providing customizable different templates. Code Style Enforcer is an excellent tool to add to your team's toolbox.


DasBlog All - Microsoft Visual Studio (Administrator)
File Edit View Refactor Project Build Debug Data Tools DevExpress Test Window Help
Debug
Any CPU
KTHXBYE
CommentViewBox.ascx
AdminNavBar.ascx
BlogRollControl.ascx
ArchivesControl.ascx
ActivityBox.ascx.cs
Start Page
Test List Editor
Toolbox
CSS Properties
Properties
Solution Explorer
newtelligence.DasBlog.Web.ActivityBox
Page_Load(object sender, System.EventArgs e)
Error List
Find Results 1
Find Symbol Results
Ready
Ln 68
Col 5
Ch 5
INS

```
    /// <summary>
    /// Summary description for WebForm1.
    /// </summary>
    public partial class ActivityBox : System.Web.UI.UserControl
    {
        protected System.Web.UI.WebControls.Label labelTitle;
        protected System.Web.UI.HtmlControls.HtmlGenericControl editNavigatorLinks;
        protected System.Web.UI.HtmlControls.HtmlGenericControl editCrossPostSites;
        protected System.Resources.ResourceManager resmgr;
        protected SiteConfig siteConfig;

        protected void Page_Load                  em.EventArgs e)
        {
            if (SiteSecurity.IsInRole("admin") == false)
            {
                Response.Redirect("~/FormatPage.aspx?path=SiteConfig/accessdenied.format.html"

            }

            //set the current page <li> tag to here
            string filePath = HttpContext.Current.Request.FilePath.ToLower();

            if (filePath.EndsWith("CrosspostReferrers.aspx".ToLower()))
            {
                crosspost.Attributes["class"]="here";
                hyperlinkCrosspostReferrers.NavigateUrl= "";
            }
            else if (filePath.EndsWith("UserClickThroughs.aspx".ToLower()))
            {
```

Prefix is wrong, should be 'm_'.
Case is wrong.
Right-click to fix.

Figure B-10

Packer for .NET — Javascript Minimizer

When creating an ASP.NET Web site, you often find yourself creating custom JavaScript files. During development, you want these files to be commented and easy to read. During production, however, every byte counts and it's nice to reduce the size of your JavaScript files with a JavaScript ''minimizer.''

Packer for .NET is a C# application that offers compression of JavaScript or simple ''minification'' by stripping comments and white space.

You'd be surprised how well these techniques work. For example, Steve Kallestad reported that a copy of the JavaScript library Prototype 1.50 was 70K before JavaScript-specific compression. It became 30K after the process, and then reached only 14K when gzip HTTP compression was applied. From 70K to 14K is a pretty significant savings.

JavaScript-specific compression does things such as renaming variables to single letters, being aware of global variable renaming vs. local variable renaming, as well as stripping unnecessary white space and comments.

Packer includes utilities for compression at both the command line and within MSBuild projects. The MSBuild targets can be added to your build process. Consequently, your integration server is continuous so you receive these benefits automatically and unobtrusively.

As an example, a JavaScript library might start out looking like this:

```
var Prototype = {
  Version: '1.5.0',
  BrowserFeatures: {
    XPath: !!document.evaluate
  },

  ScriptFragment: '(?:<script.*?>)((\n|\r|.)*?)(?:<\/script>)',
  emptyFunction: function() {},
  K: function(x) { return x }
}
```

Packed, the JavaScript might end up looking like this (as an example). . .but it will still work!

```
(c(){f 7.2q(/<\\/?[^>]+>/5a,"")}),2C:(c(){f 7.2q(P
5d(1m.5s,"9n"),"")}),9j:(c(){k 9m=P 5d(1m.5s,"9n");k 9k=P
5d(1m.5s,"ce");f(7.E(9m)||[]).1F((c(9l){f(9l.E(9k)||["",""])[1]}))}),3P
:(c(){f 7.9j().1F((c(4s){f 6A(4s)}))}),cd:(c(){k 1h=N.4f("1h");k
2V=N.cc(7);1h.63(2V);f 1h.2P}),cb:(c(){k 1h=N.4f("1h");1h.2P=7.9i();f
1h.2O[0]?(1h.2O.o>1?$A(1h.2O).2A("",(c(3Y,1G){f
3Y+1G.4j})):1h.2O[0].4j):""}),6J:(c(9h){k
E=7.4d().E(/([^?#]*)(#.*)?$/);h(!E){f{}}f
E[1].3m(9h||"&").2A({},(c(2E,Q){h((Q=Q.3m("="))[0]){k v=9g(Q[0]);k
l=Q[1]?9g(Q[1]):1b;h(2E[v]!==1b){h(2E[v].3k!=1M){2E[v]=[2E[v>}h(l){2E
[v].M(l)}}1k{2E[v]=l}}f 2E}))}),2F:(c(){f 7.3m("")})
```

There are many JavaScript minimizing libraries available; this is just one of them. However, it’s options, completeness, and integration with MSBuild that make Packer for .NET worth trying out.

Visual Studio Add-ins

‘‘If I had eight hours to chop down a tree, I’d spend six sharpening my axe.’’ — Abraham Lincoln

ASPX Edit Helper Add-In for Visual Studio

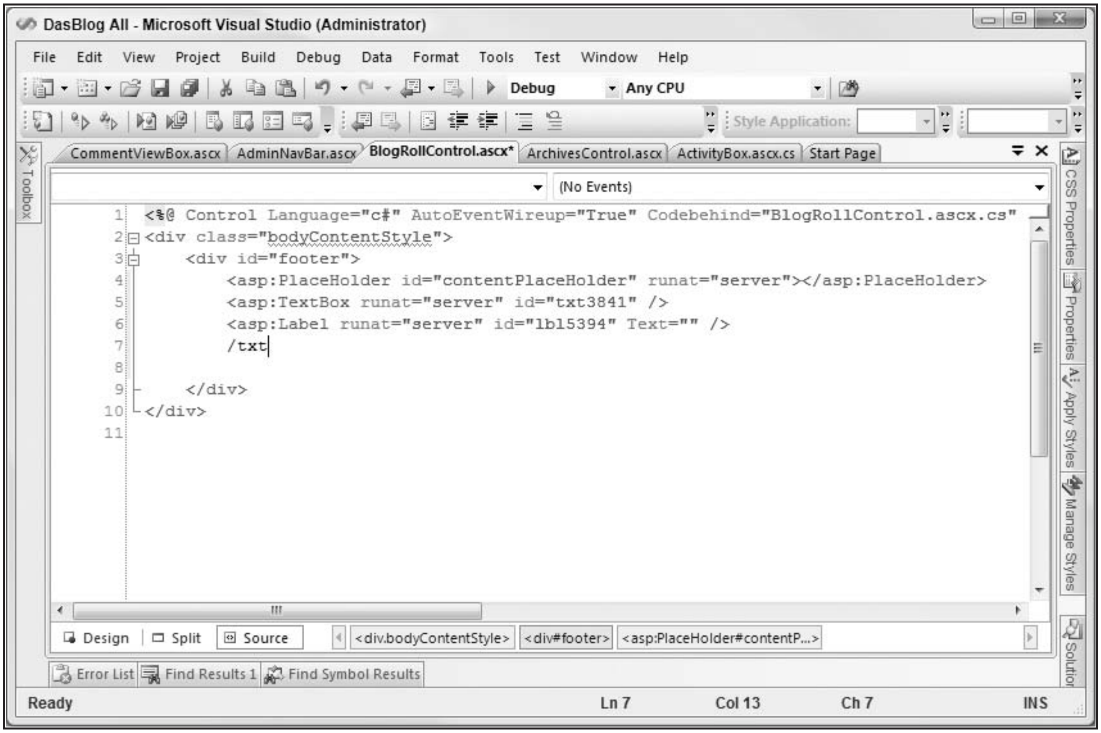
Sometimes an add-in does something so small and so simple that you might dismiss it at first glance. But when you find yourself doing the same action over and over again, you'll be thankful for the ingenuity of developers who create time savers like this little gem.

The ASPX Edit Helper does two things, and it does them well. Not everyone likes to use the visual designer in Visual Studio 2008. Some users prefer to type ASPX mark-up directly in the source view. This add-in automatically fills in `"runat="server"` and `id="randomid"` when you type a server control. It also includes short codes for automatic insertion of snippets.

For example, if you type `/lbl` and press Enter, you'll get:

```
<asp:Label runat="server" id="lbl5394" Text="" />
```

A nice touch is that the cursor will be automatically positioned within the id attribute in the random ID selected so you can immediately begin typing a new ID for the control. This thoughtful design will allow you to create complicated forms with minimal typing very quickly. Figure B-11 shows the moment just before I press Enter after typing the `/txt` short code.


DasBlog All - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Data Format Tools Test Window Help
Debug
Any CPU
Style Application:
CommentViewBox.ascx
AdminNavBar.ascx
BlogRollControl.ascx*
ArchivesControl.ascx
ActivityBox.ascx.cs
Start Page
(No Events)
Toolbox
CSS Properties
Properties
Apply Styles
Manage Styles
Solution
Design
Split
Source
<div.bodyContentStyle>
<div#footer>
<asp:PlaceHolder#contentP...>
Error List
Find Results 1
Find Symbol Results
Ready
Ln 7
Col 13
Ch 7
INS

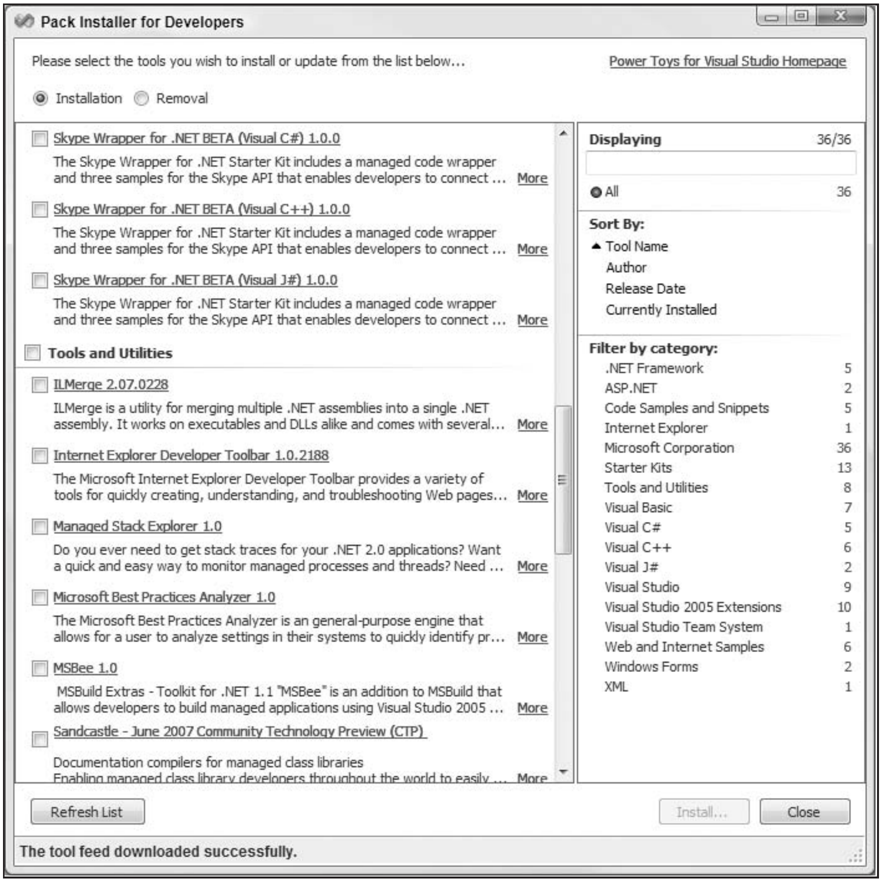
```
<%@ Control Language="c#" AutoEventWireup="True" Codebehind="BlogRollControl.ascx.cs"
<div class="bodyContentStyle">
    <div id="footer">
        <asp:PlaceHolder id="contentPlaceHolder" runat="server"></asp:PlaceHolder>
        <asp:TextBox runat="server" id="txt3841" />
        <asp:Label runat="server" id="lbl5394" Text="" />
        /txt

    </div>
</div>
```

Figure B-11

Installation is easy. Just unzip into your Visual Studio Add-ins folder. As an aside, the authors also include source code, which is useful if you're interested in writing your own Add-in.

Power Toys Pack Installer

What's better than a useful Visual Studio Add-in? Why, an automatic installer that downloads a list of useful add-ins and lets you select them and install at once, that's what! The Power Toys Pack Installer, shown in Figure B-12, is an open source CodePlex project that downloads a feed full of at least three dozen projects, samples, starter kits and extensions that can be selected and installed *en masse*.



Pack Installer for Developers

Please select the tools you wish to install or update from the list below...

Power Toys for Visual Studio Homepage

● Installation ○ Removal

☐ Skype Wrapper for .NET BETA (Visual C#) 1.0.0
The Skype Wrapper for .NET Starter Kit includes a managed code wrapper and three samples for the Skype API that enables developers to connect ... More

☐ Skype Wrapper for .NET BETA (Visual C++) 1.0.0
The Skype Wrapper for .NET Starter Kit includes a managed code wrapper and three samples for the Skype API that enables developers to connect ... More

☐ Skype Wrapper for .NET BETA (Visual J#) 1.0.0
The Skype Wrapper for .NET Starter Kit includes a managed code wrapper and three samples for the Skype API that enables developers to connect ... More

☐ **Tools and Utilities**

☐ ILMerge 2.07.0228
ILMerge is a utility for merging multiple .NET assemblies into a single .NET assembly. It works on executables and DLLs alike and comes with several... More

☐ Internet Explorer Developer Toolbar 1.0.2188
The Microsoft Internet Explorer Developer Toolbar provides a variety of tools for quickly creating, understanding, and troubleshooting Web pages... More

☐ Managed Stack Explorer 1.0
Do you ever need to get stack traces for your .NET 2.0 applications? Want a quick and easy way to monitor managed processes and threads? Need ... More

☐ Microsoft Best Practices Analyzer 1.0
The Microsoft Best Practices Analyzer is an general-purpose engine that allows for a user to analyze settings in their systems to quickly identify pr... More

☐ MSBee 1.0
MSBuild Extras - Toolkit for .NET 1.1 "MSBee" is an addition to MSBuild that allows developers to build managed applications using Visual Studio 2005 ... More

☐ Sandcastle - June 2007 Community Technology Preview (CTP)
Documentation compilers for managed class libraries
Enabling managed class library developers throughout the world to easily ... More

Displaying 36/36

● All 36

Sort By:
▲ Tool Name
Author
Release Date
Currently Installed

Filter by category:

.NET Framework	5
ASP.NET	2
Code Samples and Snippets	5
Internet Explorer	1
Microsoft Corporation	36
Starter Kits	13
Tools and Utilities	8
Visual Basic	7
Visual C#	5
Visual C++	6
Visual J#	2
Visual Studio	9
Visual Studio 2005 Extensions	10
Visual Studio Team System	1
Web and Internet Samples	6
Windows Forms	2
XML	1

The tool feed downloaded successfully.

Figure B-12

The selection is cherry-picked from projects and downloads all over Microsoft. Some highlights include Microsoft. NET Interfaces for Skype and starter kits for creating your own Shareware. Developer tools include ILMerge, a utility for merging multiple .NET assemblies into a single assembly, the Internet Explorer Developer Toolbar mentioned earlier in this chapter, a Managed Stack Explorer for investigating application hangs, and XML Notepad 2007, an experimental interface for browsing and editing XML documents. There are also choice Add-Ins such as the VS Source Outliner, giving you a tree view of your project's member and types for use as an alternative navigational method. A few tools aren't ready for Visual Studio 2008, but I expect to see them updated, given community pressure.

Extending ASP.NET

"Oh man! :-) I have shoot into my foot myself ;-) Sorry!" — matz."

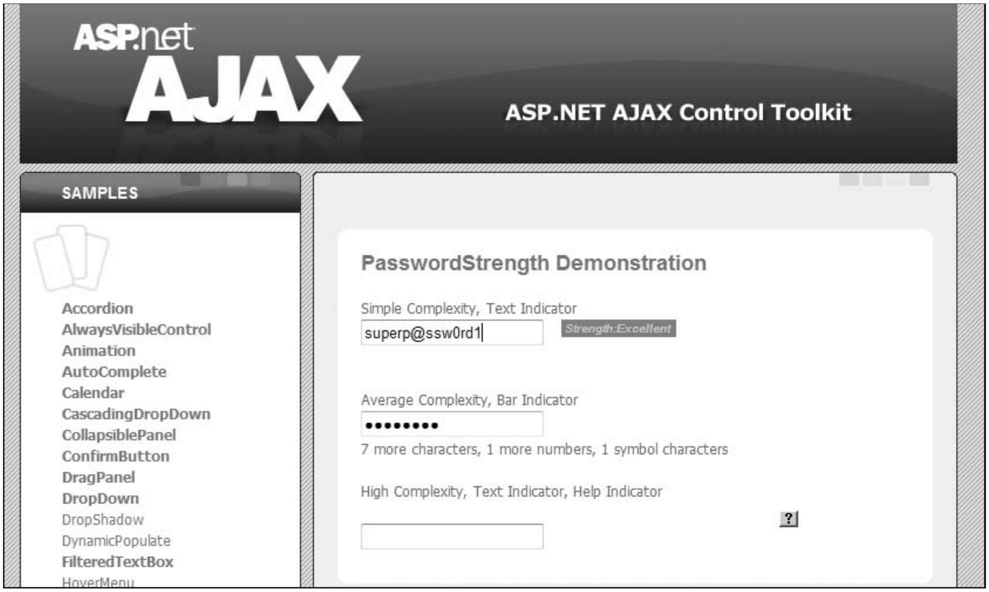
ASP.NET AJAX Control Toolkit

The AJAX Control Toolkit is a collaboration between Microsoft and the larger ASP.NET community. Its goal was to provide the largest collection of Web client components available. It includes excellent examples if you want to learn how to write ASP.NET Ajax yourself, and then it gives you the opportunity to give back and have your code shared within the community.

There are literally dozens of controls that build on and extend the ASP.NET Ajax framework. Some of the controls are simple and provide those nice "little touches" such as drop shadows, rounded corners, watermarks, and animations. Others provide highly functional controls such as calendars, popups, and sliders.

Complete source is available for all the controls so that they can be extended and improved by you. These controls are more than just samples; they are complete and ready to be used in your applications.

There's a complete demo site available at `http://ajax.asp.net/ajaxtoolkit/` showcasing examples of each control so you can try each one to see if it meets your needs, as illustrated in Figure B-13, for example.


ASP.net
AJAX
ASP.NET AJAX Control Toolkit
SAMPLES
Accordion
AlwaysVisibleControl
Animation
AutoComplete
Calendar
CascadingDropDown
CollapsiblePanel
ConfirmButton
DragPanel
DropDown
DropShadow
DynamicPopulate
FilteredTextBox
HoverMenu
?

PasswordStrength Demonstration

Simple Complexity, Text Indicator

superp@ssw0rd1 Strength:Excellent

Average Complexity, Bar Indicator

•••••••••

7 more characters, 1 more numbers, 1 symbol characters

High Complexity, Text Indicator, Help Indicator

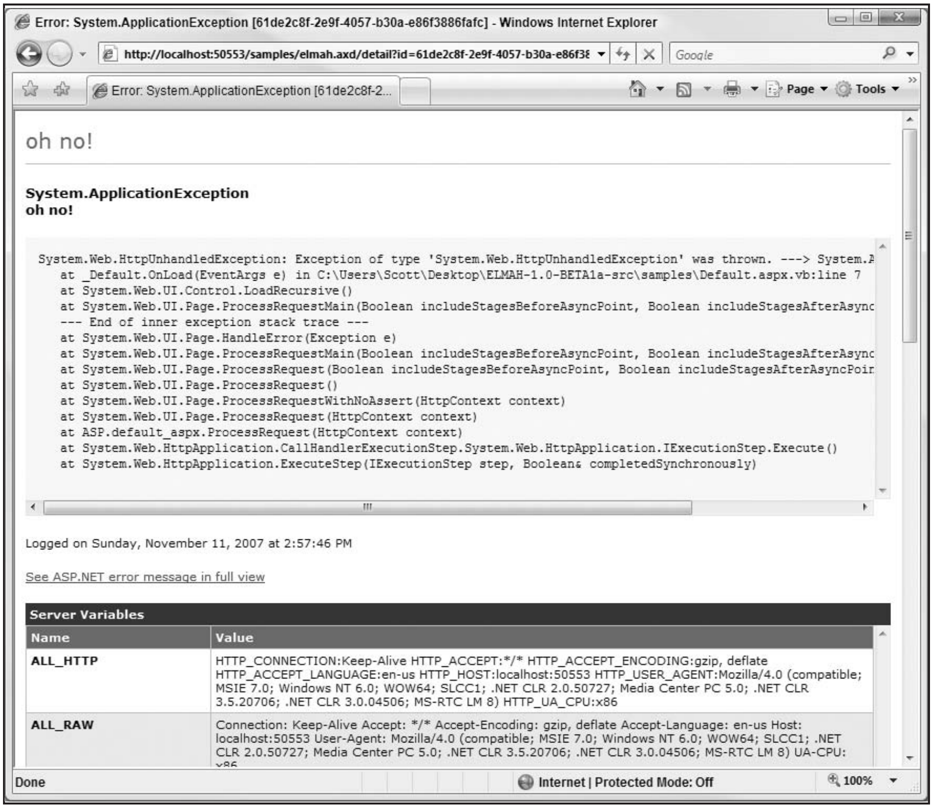
Figure B-13

Note that there are two versions one for Visual Studio 2005 and one for Visual Studio 2008. Features that are specific to the 2008 version include "reference tags" for toolkit JavaScript files providing more complete JavaScript IntelliSense support within Visual Studio.

Atif Aziz's ELMAH — Error Logging Modules and Handlers

Troubleshooting errors and unhandled exceptions in your applications can be a full-time job. Rather than writing your own custom global exception handlers every time, consider looking at the ELMAH (Error Logging Modules And Handlers) from Atif Aziz. It's a very flexible application-wide error logging facility with pluggable extension points to the interfaces at nearly every location. You can even configure it in your application without re-compilation or even redeployment. Simply modify your `web.config` to include the error logging modules and handlers, and then you'll receive a single Web page to remotely review the entire log of unhandled exceptions.

ELMAH captures so much information about exceptions that it can reconstitute the original ''yellow screen of death'' that ASP.NET would have generated given an exception, even if customErrors was turned off. It's almost like TiVo for your exceptions! Figure B-14 shows ELMAH, providing a developer's view, including all the details you might need to debug this error.


Error: System.ApplicationException [61de2c8f-2e9f-4057-b30a-e86f3886fafc] - Windows Internet Explorer
http://localhost:50553/samples/elmah.axd/detail?id=61de2c8f-2e9f-4057-b30a-e86f3
Google
Error: System.ApplicationException [61de2c8f-2...
Page
Tools
Done
Internet | Protected Mode: Off
100%

oh no!

System.ApplicationException
oh no!

```
System.Web.HttpUnhandledException: Exception of type 'System.Web.HttpUnhandledException' was thrown. ---> System.A
   at _Default.OnLoad(EventArgs e) in C:\Users\Scott\Desktop\ELMAH-1.0-BETA1a-src\samples\Default.aspx.vb:line 7
   at System.Web.UI.Control.LoadRecursive()
   at System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean includeStagesAfterAsync
   --- End of inner exception stack trace ---
   at System.Web.UI.Page.HandleError(Exception e)
   at System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean includeStagesAfterAsync
   at System.Web.UI.Page.ProcessRequest(Boolean includeStagesBeforeAsyncPoint, Boolean includeStagesAfterAsyncPoir
   at System.Web.UI.Page.ProcessRequest()
   at System.Web.UI.Page.ProcessRequestWithNoAssert(HttpContext context)
   at System.Web.UI.Page.ProcessRequest(HttpContext context)
   at ASP.default_aspx.ProcessRequest(HttpContext context)
   at System.Web.HttpApplication.CallHandlerExecutionStep.System.Web.HttpApplication.IExecutionStep.Execute()
   at System.Web.HttpApplication.ExecuteStep(IExecutionStep step, Boolean& completedSynchronously)
```

Logged on Sunday, November 11, 2007 at 2:57:46 PM

See ASP.NET error message in full view

Server Variables

Name	Value
ALL_HTTP	HTTP_CONNECTION:Keep-Alive HTTP_ACCEPT:*/* HTTP_ACCEPT_ENCODING:gzip, deflate HTTP_ACCEPT_LANGUAGE:en-us HTTP_HOST:localhost:50553 HTTP_USER_AGENT:Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; WOW64; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.5.20706; .NET CLR 3.0.04506; MS-RTC LM 8) HTTP_UA_CPU:x86
ALL_RAW	Connection: Keep-Alive Accept: */* Accept-Encoding: gzip, deflate Accept-Language: en-us Host: localhost:50553 User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; WOW64; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.5.20706; .NET CLR 3.0.04506; MS-RTC LM 8) UA-CPU: x86

Figure B-14

Another clever feature is an RSS feed that shows the last 15 years from your log. This flexible tool is open source and the recent beta includes support for medium trust environments. You can plug in SQL Server or use an XML file to manage your error logs. I highly recommend you take the time to learn about ELMAH.

Helicon's ISAPI_Rewrite

Users of the Apache Web server sing the praises of the power of `mod_rewrite`, their URL rewriting mechanism. IIS users have this available to them in the form of the ISAPI Rewrite module from Helicon. It's incredibly fast because it's written in pure C. It integrates nicely with ASP.NET because URLs are rewritten before ASP.NET realizes anything has happened.

Because it uses regular expressions, it can initially be very frustrating due to its terse syntax. However, if you are patient, it can be an incredibly powerful tool for your tool belt.

I recently discovered that there were a dozen ways to visit my blog that all lead to the same location. This was a confusing Google because it appeared that my blog had multiple addresses. I wanted not only to canonicalize my URL but also send back a 301 HTTP redirect to search indexes, thereby raising my standing within the search by appearing to have only one official URL.

For example, all of these links were valid ways to reach my blog:

- ☐ `www.hanselman.com/blog/`
- ☐ `www.hanselman.com/blog/default.aspx`
- ☐ `www.hanselman.com/blog`
- ☐ `http://hanselman.com/blog/`
- ☐ `http://hanselman.com/blog/default.aspx`
- ☐ `http://hanselman.com/blog`
- ☐ `www.hanselman.com/blog/Default.aspx`
- ☐ `www.computerzen.com/blog/`
- ☐ `www.computerzen.com`
- ☐ `http://computerzen.com/blog/`
- ☐ `http://computerzen.com/`

Notice that there's a difference between a trailing slash and no trailing slash in the eyes of a search engine. Using ISAPI Rewrite, I created this rather terse but very effective configuration file:

```
[ISAPI_Rewrite]
RewriteRule /blog/default\.aspx http\://www.hanselman.com/blog/ [I,RP]
RewriteCond Host: ^hanselman\.com
RewriteRule (.*) http\://www.hanselman.com$1 [I,RP]
RewriteCond Host: ^computerzen\.com
RewriteRule (.*) http\://www.hanselman.com$1 [I,RP]
RewriteCond Host: ^www.computerzen\.com
RewriteRule (.*) http\://www.hanselman.com/blog/ [I,RP]
```

The I and RP at the end of the line indicate that this match is case insensitive and the redirect should be permanent rather than temporary. The rules that include a $1 at the end of line cause the expression to include any path after the domain name. This allows the rule to apply site-wide and provides these benefits to every single page on my site. It's powerful and that's worth your time.

General Purpose Developer Tools

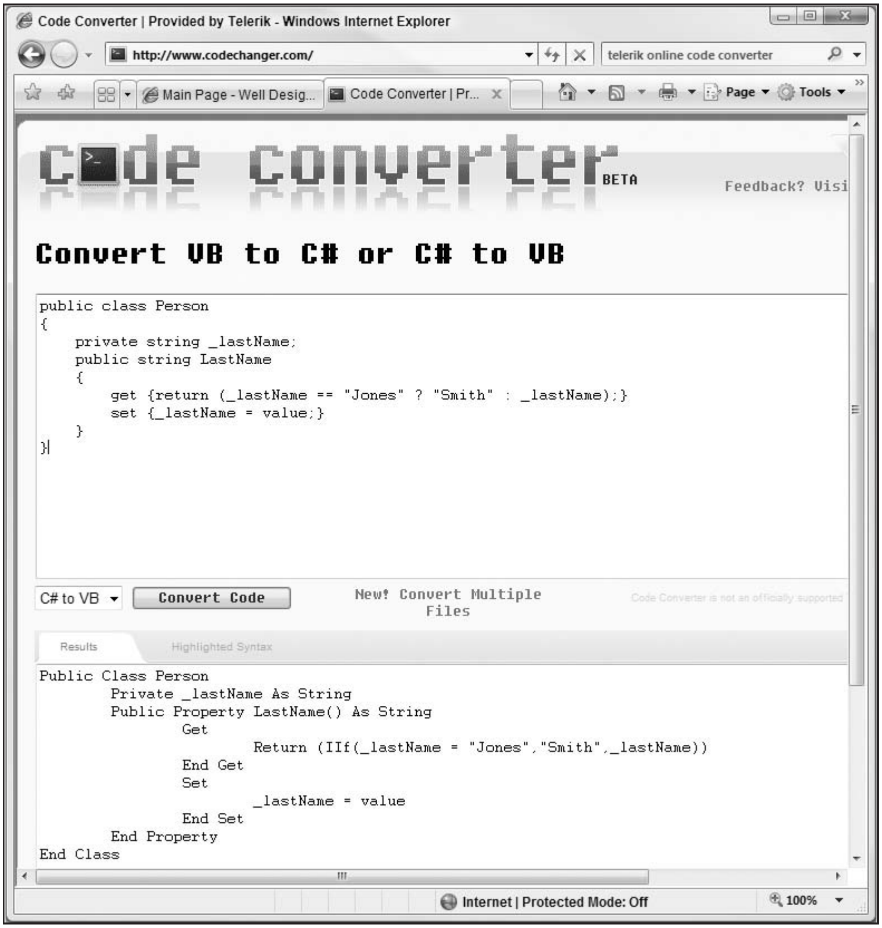
> "If you get the dirty end of the stick, sharpen it and turn it into a useful tool" — Colin Powell

Telerik's Online Code Converter

Creating samples that should appear in both C# and Visual Basic can be very tedious without the assistance of something like Telerik's CodeChanger.com.

While it's not an officially supported tool, this little application will definitely get you 80 percent of the way when converting between Visual Basic and C#.

It also understands a surprising number of rather obscure syntaxes, as shown in Figure B-15, where I tried to convert an immediate `if` from C#'s `?:` syntax to VB's `IIf` syntax. It's not only useful for the writer, and blog author, but also anyone who's trying to switch projects between the two languages.


Code Converter | Provided by Telerik - Windows Internet Explorer
http://www.codechanger.com/
telerik online code converter
Main Page - Well Desig...
Code Converter | Pr...
Page
Tools
code converter BETA
Feedback? Visi
Convert VB to C# or C# to VB
C# to VB
Convert Code
New! Convert Multiple Files
Code Converter is not an officially supported
Results
Highlighted Syntax
Internet | Protected Mode: Off
100%

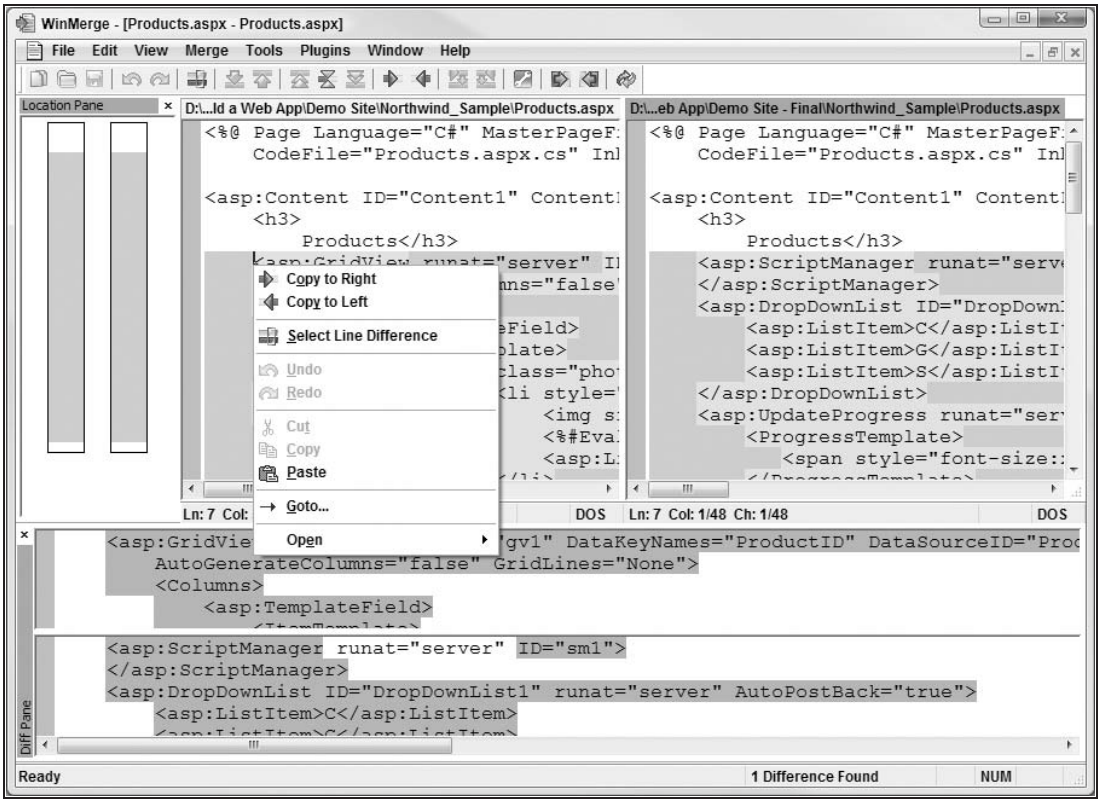
```
public class Person
{
    private string _lastName;
    public string LastName
    {
        get {return (_lastName == "Jones" ? "Smith" : _lastName);}
        set {_lastName = value;}
    }
}
```

```
Public Class Person
        Private _lastName As String
        Public Property LastName() As String
                Get
                        Return (IIf(_lastName = "Jones","Smith",_lastName))
                End Get
                Set
                        _lastName = value
                End Set
        End Property
End Class
```

Figure B-15

WinMerge and Differencing Tools

Everyone has their favorite merge tool. Whether yours is WinMerge (Figure B-16), or Beyond Compare, or the old standby WinDiff, just make sure that you have one in your list of tools that you're very familiar with. When managing large numbers of changes across large numbers of individuals on software development teams, a good merge tool can help you untangle even the most complicated conflicting checkins.


WinMerge - [Products.aspx - Products.aspx]
File
Edit
View
Merge
Tools
Plugins
Window
Help
Location Pane
D:\...ld a Web App\Demo Site\Northwind_Sample\Products.aspx
D:\...eb App\Demo Site - Final\Northwind_Sample\Products.aspx
Copy to Right
Copy to Left
Select Line Difference
Undo
Redo
Cut
Copy
Paste
Goto...
Open
ns="false'
eField>
plate>
class="pho
<li style='
<img s
<%#Eva
<asp:L
Ln: 7 Col:
DOS
Ln: 7 Col: 1/48 Ch: 1/48
DOS
Diff Pane
Ready
1 Difference Found
NUM

```
<%@ Page Language="C#" MasterPageF
    CodeFile="Products.aspx.cs" Inl

<asp:Content ID="Content1" Content
    <h3>
        Products</h3>
    <asp:GridView runat="server" I
```

```
<%@ Page Language="C#" MasterPageF
    CodeFile="Products.aspx.cs" Inl

<asp:Content ID="Content1" Content
    <h3>
        Products</h3>
    <asp:ScriptManager runat="serv
    </asp:ScriptManager>
    <asp:DropDownList ID="DropDown
        <asp:ListItem>C</asp:ListI
        <asp:ListItem>G</asp:ListI
        <asp:ListItem>S</asp:ListI
    </asp:DropDownList>
    <asp:UpdateProgress runat="ser
        <ProgressTemplate>
            <span style="font-size::
```

```
<asp:GridVie          "gv1" DataKeyNames="ProductID" DataSourceID="Proc
    AutoGenerateColumns="false" GridLines="None">
    <Columns>
        <asp:TemplateField>
```

```
<asp:ScriptManager runat="server" ID="sm1">
</asp:ScriptManager>
<asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="true">
    <asp:ListItem>C</asp:ListItem>
```

Figure B-16

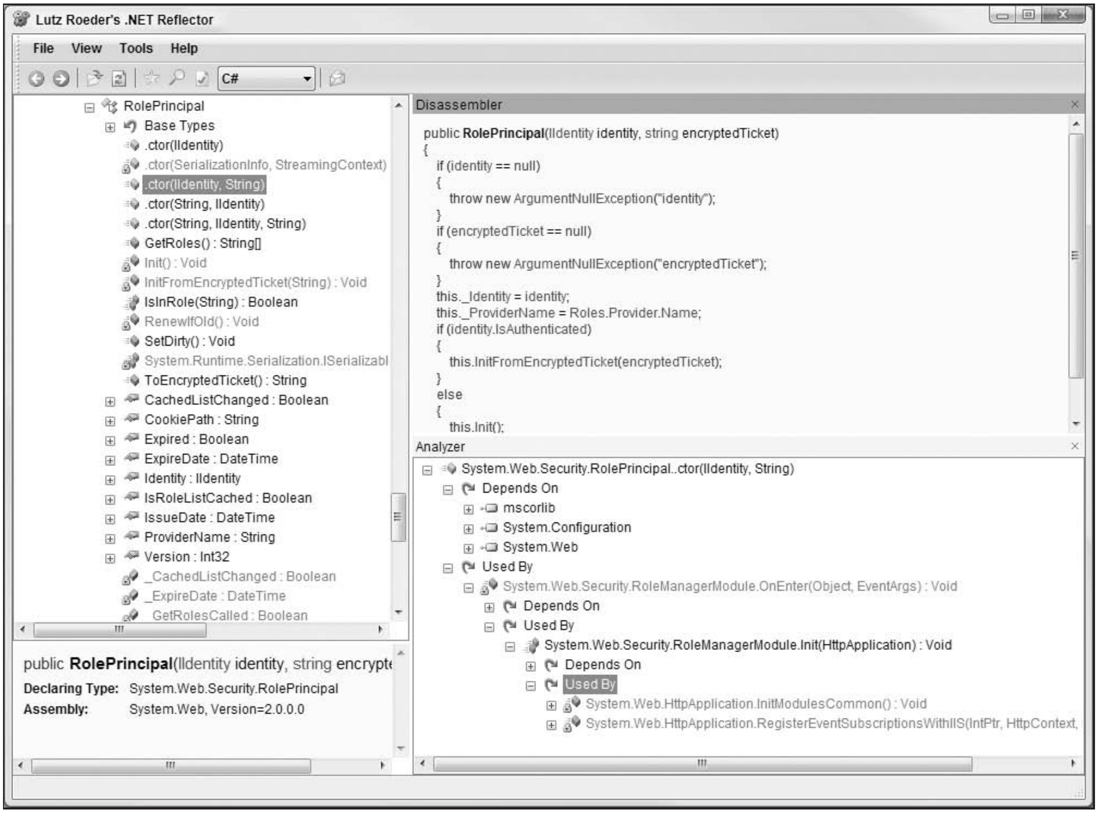
A number of different plug-ins are available for WinMerge that extend its functionality to include comparison of Word and Excel documents and XML files.

Other highly recommended merge tools include Beyond Compare from Scooter Software and DiffMerge from SourceGear. Each of these three tools integrates with Windows Explorer, so the comparing files are as easy as a right-click.

Reflector

If you're not using Reflector, your .NET developer experience is lesser for it. Reflector is an object browser, decompiler, help system, powerful plug-in host, and incredible learning tool. This tiny utility from Microsoft developer Lutz Roeder is consistently listed as the number one most indispensable tool available to the .NET developer after Visual Studio.

Reflector is amazing because it not only gives you a representation of the programmer's intent by transforming IL back into C# or VB, but it includes analysis tools that help you visualize dependencies between methods in the .NET Base Class Library and within your code or any third party code. In Figure B-17, you can see not only a C# representation of the code inside System. RolePrincipal, but more importantly the methods that use it within the framework. You can continue on as deep as you want within the theoretical call stack.


Lutz Roeder's .NET Reflector
File
View
Tools
Help
C#
RolePrincipal
Base Types
.ctor(IIdentity)
.ctor(SerializationInfo, StreamingContext)
.ctor(IIdentity, String)
.ctor(String, IIdentity)
.ctor(String, IIdentity, String)
GetRoles() : String[]
Init() : Void
InitFromEncryptedTicket(String) : Void
IsInRole(String) : Boolean
RenewIfOld() : Void
SetDirty() : Void
System.Runtime.Serialization.ISerializabl
ToEncryptedTicket() : String
CachedListChanged : Boolean
CookiePath : String
Expired : Boolean
ExpireDate : DateTime
Identity : IIdentity
IsRoleListCached : Boolean
IssueDate : DateTime
ProviderName : String
Version : Int32
_CachedListChanged : Boolean
_ExpireDate : DateTime
_GetRolesCalled : Boolean
Disassembler
Analyzer
System.Web.Security.RolePrincipal..ctor(IIdentity, String)
Depends On
mscorlib
System.Configuration
System.Web
Used By
System.Web.Security.RoleManagerModule.OnEnter(Object, EventArgs) : Void
Depends On
Used By
System.Web.Security.RoleManagerModule.Init(HttpApplication) : Void
Depends On
Used By
System.Web.HttpApplication.InitModulesCommon() : Void
System.Web.HttpApplication.RegisterEventSubscriptionsWithIIS(IntPtr, HttpContext,

```
public RolePrincipal(IIdentity identity, string encryptedTicket)
{
   if (identity == null)
   {
      throw new ArgumentNullException("identity");
   }
   if (encryptedTicket == null)
   {
      throw new ArgumentNullException("encryptedTicket");
   }
   this._Identity = identity;
   this._ProviderName = Roles.Provider.Name;
   if (identity.IsAuthenticated)
   {
      this.InitFromEncryptedTicket(encryptedTicket);
   }
   else
   {
      this.Init();
```

public **RolePrincipal**(IIdentity identity, string encrypte

Declaring Type: System.Web.Security.RolePrincipal

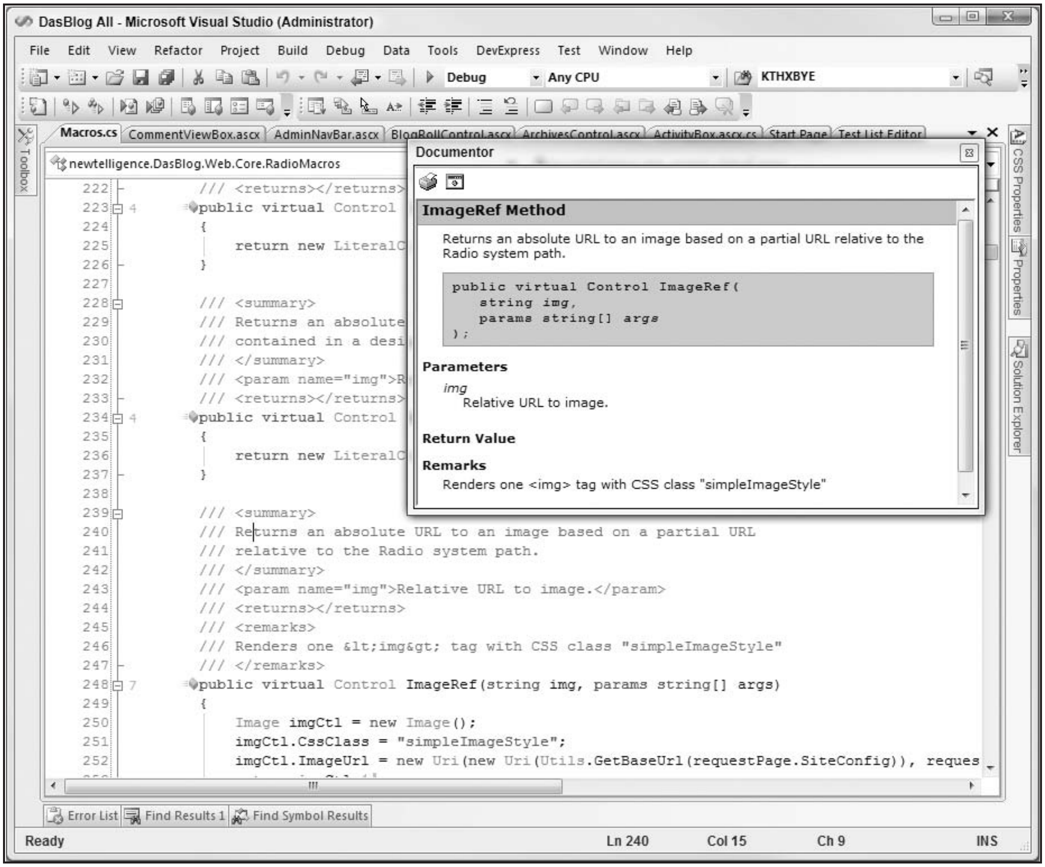
Assembly: System.Web, Version=2.0.0.0

Figure B-17

While Reflector's decompilation abilities may become less useful with the release of the Base Class Library source code under the Microsoft Reference License, its abilities as an object browser and its vibrant plug-in community will keep this tool on the top shelf for years to come.

CR_Documentor

CR_Documentor (Figure B-18) is another free plug-in that uses the DxCore extension technology from DevExpress. This tool is a collaboration between developer Travis Illig and Reflector author Lutz Roeder.


DasBlog All - Microsoft Visual Studio (Administrator)
File Edit View Refactor Project Build Debug Data Tools DevExpress Test Window Help
Debug
Any CPU
KTHXBYE
Toolbox
Macros.cs
CommentViewBox.ascx
AdminNavBar.ascx
BlogRollControl.ascx
ArchivesControl.ascx
ActivityBox.ascx.cs
Start Page
Test List Editor
newtelligence.DasBlog.Web.Core.RadioMacros
Documentor
CSS Properties
Properties
Solution Explorer
Error List
Find Results 1
Find Symbol Results
Ready
Ln 240
Col 15
Ch 9
INS

```
        /// <returns></returns>
        public virtual Control
        {
            return new LiteralC
        }

        /// <summary>
        /// Returns an absolute
        /// contained in a desi
        /// </summary>
        /// <param name="img">R
        /// <returns></returns>
        public virtual Control
        {
            return new LiteralC
        }

        /// <summary>
        /// Returns an absolute URL to an image based on a partial URL
        /// relative to the Radio system path.
        /// </summary>
        /// <param name="img">Relative URL to image.</param>
        /// <returns></returns>
        /// <remarks>
        /// Renders one &lt;img&gt; tag with CSS class "simpleImageStyle"
        /// </remarks>
        public virtual Control ImageRef(string img, params string[] args)
        {
            Image imgCtl = new Image();
            imgCtl.CssClass = "simpleImageStyle";
            imgCtl.ImageUrl = new Uri(new Uri(Utils.GetBaseUrl(requestPage.SiteConfig)), reques
```

ImageRef Method

Returns an absolute URL to an image based on a partial URL relative to the Radio system path.

```
public virtual Control ImageRef(
   string img,
   params string[] args
);
```

Parameters

img
Relative URL to image.

Return Value

Remarks

Renders one <img> tag with CSS class "simpleImageStyle"

Figure B-18

This add-on allows you to see a preview of your XML document comments in real time as you edit them within your source code in Visual Studio.

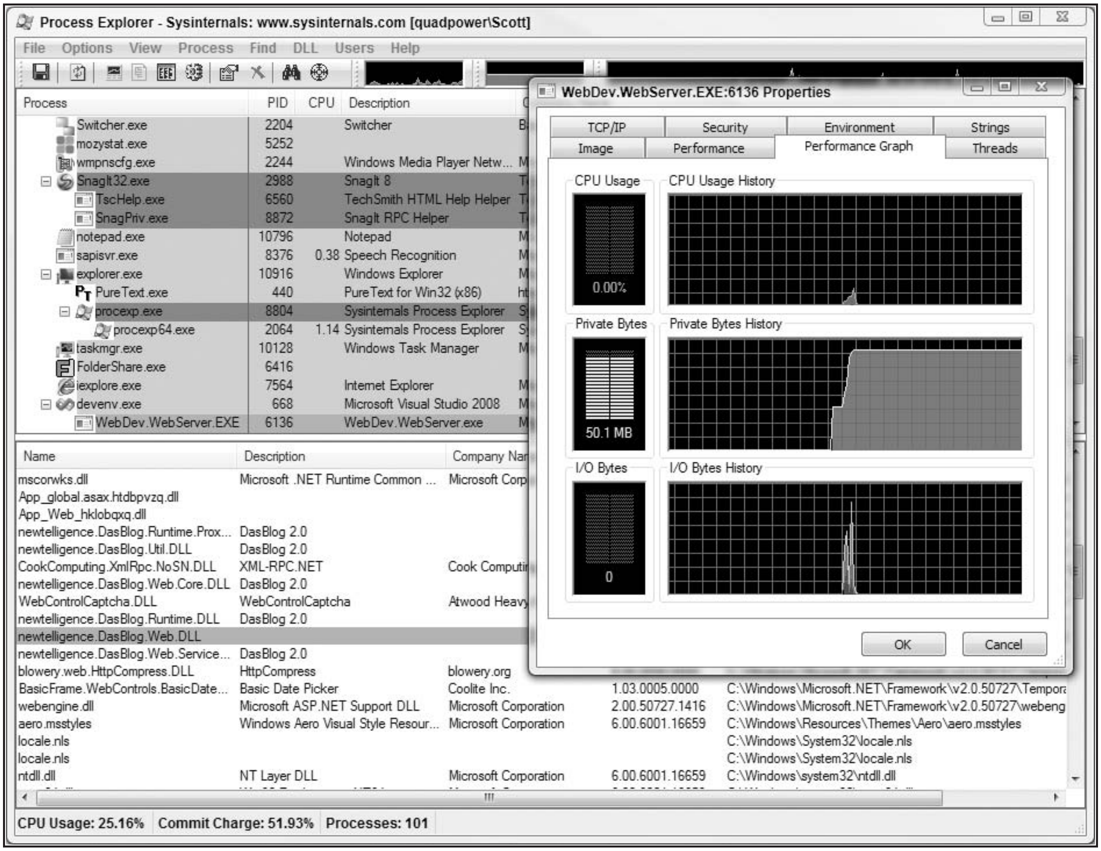
CR_Documentor lets you choose your level of compliance including support for the now-defunct-but-still-useful NDoc project as well as Microsoft specific tags for use with the Microsoft ''Sandcastle'' Documentation building suite of tools.

CR_Documentor also includes context menu support with snippets and helpful templates making it much easier to visualize complicated documentation. XML documentation source code provides all the tools and tags needed to make MSDN-quality help. CR_Documentor provides that last missing piece in the form of excellent and accurate visualization. It's indispensable if you intend on building compiled help files from your source code.

Process Explorer

Last, but unquestionably not least, is Process Explorer from Mark Russinovich. To call it ''Task Manager on steroids'' would not even begin to do it justice. Process Explorer puts windows itself under a microscope by allowing you to peer inside your active processes, their threads, and the environment to get a clearer understanding about what is actually going on. Advanced and detailed use of this tool, along with the entire SysInternals Suite of Tools, should be required for all developers.

In Figure B-19, I'm looking at the properties dialog box of an application running under the Visual Studio 2008 Web Server while a debugging session in process. I can see the process tree, the DLLs loaded into the Web server's process, their versions, and their paths, making this an excellent tool for debugging assembly loading and versioning issues.


Process Explorer - Sysinternals: www.sysinternals.com [quadpower\Scott]
File
Options
View
Process
Find
DLL
Users
Help
WebDev.WebServer.EXE:6136 Properties
TCP/IP
Security
Environment
Strings
Image
Performance
Performance Graph
Threads
CPU Usage
CPU Usage History
0.00%
Private Bytes
Private Bytes History
50.1 MB
I/O Bytes
I/O Bytes History
0
OK
Cancel

Process	PID	CPU	Description
Switcher.exe	2204		Switcher
mozystat.exe	5252		
wmpnscfg.exe	2244		Windows Media Player Netw...
SnagIt32.exe	2988		SnagIt 8
TscHelp.exe	6560		TechSmith HTML Help Helper
SnagPriv.exe	8872		SnagIt RPC Helper
notepad.exe	10796		Notepad
sapisvr.exe	8376	0.38	Speech Recognition
explorer.exe	10916		Windows Explorer
PureText.exe	440		PureText for Win32 (x86)
procexp.exe	8804		Sysinternals Process Explorer
procexp64.exe	2064	1.14	Sysinternals Process Explorer
taskmgr.exe	10128		Windows Task Manager
FolderShare.exe	6416		
iexplore.exe	7564		Internet Explorer
devenv.exe	668		Microsoft Visual Studio 2008
WebDev.WebServer.EXE	6136		WebDev.WebServer.exe

Name	Description	Company Nar		
mscorwks.dll	Microsoft .NET Runtime Common ...	Microsoft Corp		
App_global.asax.htdbpvzq.dll				
App_Web_hklobqxq.dll				
newtelligence.DasBlog.Runtime.Prox...	DasBlog 2.0			
newtelligence.DasBlog.Util.DLL	DasBlog 2.0			
CookComputing.XmlRpc.NoSN.DLL	XML-RPC.NET	Cook Computi		
newtelligence.DasBlog.Web.Core.DLL	DasBlog 2.0			
WebControlCaptcha.DLL	WebControlCaptcha	Atwood Heavy		
newtelligence.DasBlog.Runtime.DLL	DasBlog 2.0			
newtelligence.DasBlog.Web.DLL				
newtelligence.DasBlog.Web.Service...	DasBlog 2.0			
blowery.web.HttpCompress.DLL	HttpCompress	blowery.org		
BasicFrame.WebControls.BasicDate...	Basic Date Picker	Coolite Inc.	1.03.0005.0000	C:\Windows\Microsoft.NET\Framework\v2.0.50727\Tempor
webengine.dll	Microsoft ASP.NET Support DLL	Microsoft Corporation	2.00.50727.1416	C:\Windows\Microsoft.NET\Framework\v2.0.50727\webeng
aero.msstyles	Windows Aero Visual Style Resour...	Microsoft Corporation	6.00.6001.16659	C:\Windows\Resources\Themes\Aero\aero.msstyles
locale.nls				C:\Windows\System32\locale.nls
locale.nls				C:\Windows\System32\locale.nls
ntdll.dll	NT Layer DLL	Microsoft Corporation	6.00.6001.16659	C:\Windows\system32\ntdll.dll

CPU Usage: 25.16% Commit Charge: 51.93% Processes: 101

Figure B-19

Summary

Having the right tools can mean the difference between a week of time spent in your head against the wall versus five minutes of quick analysis in debugging. The right tool can mean the difference between a tedious and keyboard-heavy code slogging or refactoring session that is actually pleasant. I encourage you to try each of the tools listed here as well as to explore the ecosystem of available tools to find those that make your development experience not just more productive but more enjoyable.

C

Silverlight

Silverlight is a lightweight browser plug-in from Microsoft that runs not only on all major browsers on Windows, but also on the Mac. Silverlight 1.0 was released in September of 2007. Many books have been written already on the subject. And now, Silverlight 2.0 is fast on the heels of version 1.0. Microsoft also announced ‘‘Moonlight,’’ a partnership with Novell and the Open Source Mono Project to bring Silverlight-like technology to Linux — you already know this, don’t you?

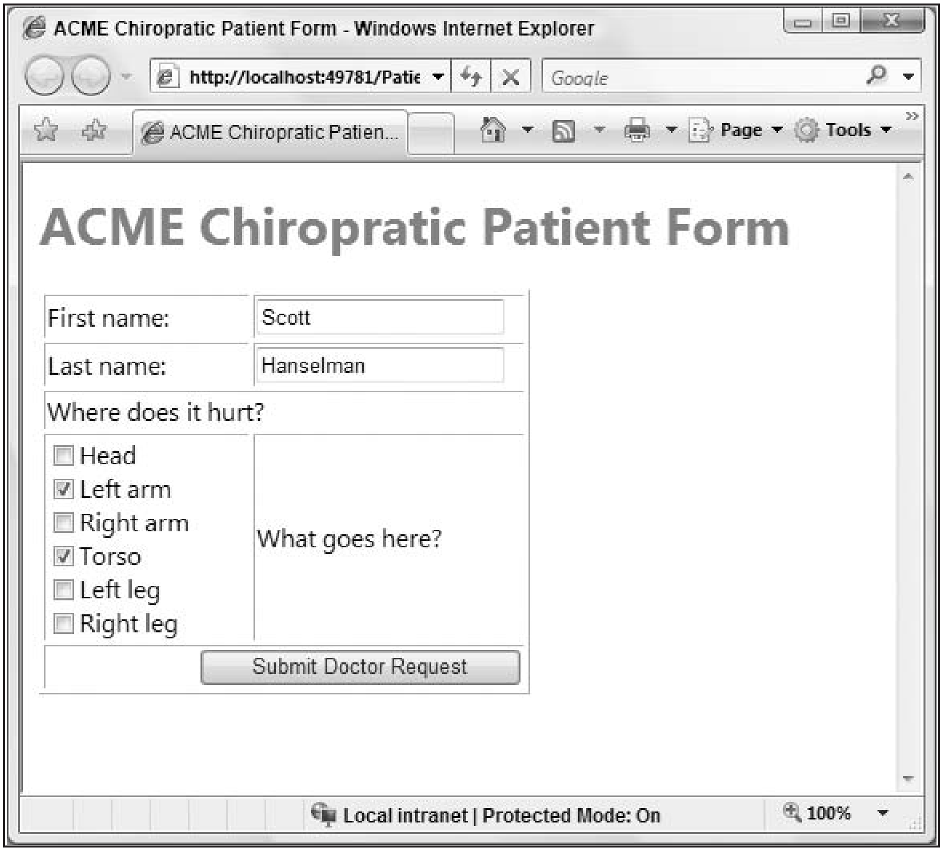
It is outside the scope of this book to give more than an introduction to Silverlight. Rather than just offering the standard ‘‘Hello World’’ example, however, we thought we’d try something more pragmatic. As this is a book on Professional ASP.NET, why not explore the scenario that a typical ASP.NET programmer (not a designer or an artist) would find herself in? Along the way you’ll learn about Silverlight and its capabilities, and perhaps it will inspire you to include Silverlight in your applications. You should read books such as *Silverlight 1.0* (Wiley Publishing, Inc., 2007) to familiarize yourself with this deceptively deep new technology.

Thank you to Eilon Lipton for his help with this application!

Extending ASP.NET Apps with Silverlight

Let’s begin with a theoretical application for patients signing into a chiropractic clinic. They need to sign in and indicate to the doctor what part of their body hurts as seen in Figure C-1. This is a very typical application using Web Controls to present a form and collect the feedback via a Form POST.

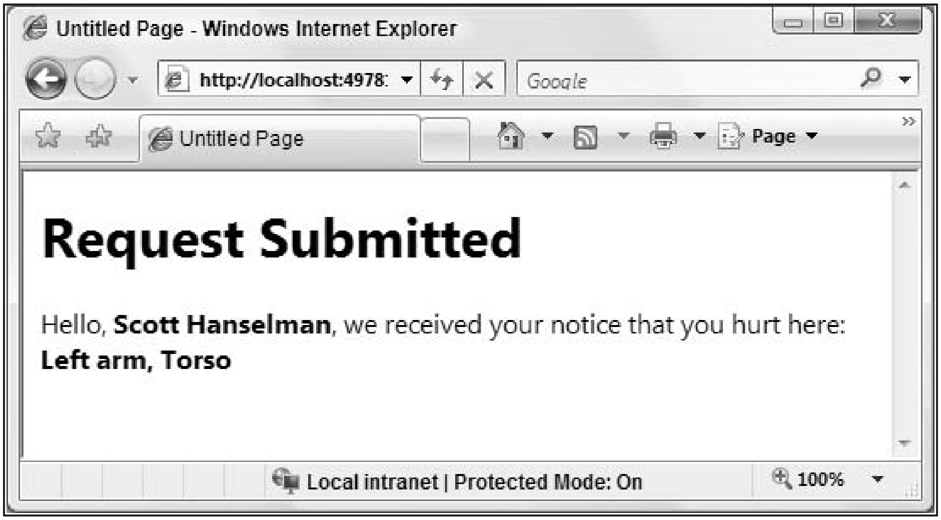
The page uses conventional ASP.NET techniques including validation controls. One interesting thing to note is that this page’s Submit button sends its data back to another page using a Cross Page PostBack. The page that receives the Form POST is seen in Figure C-2.


ACME Chiropratic Patient Form - Windows Internet Explorer
http://localhost:49781/Patie
Google
ACME Chiropratic Patien...
Page
Tools
Local intranet | Protected Mode: On
100%

ACME Chiropratic Patient Form

First name:	Scott
Last name:	Hanselman
Where does it hurt?	
☐ Head ☑ Left arm ☐ Right arm ☑ Torso ☐ Left leg ☐ Right leg	What goes here?
	Submit Doctor Request

Figure C-1


Untitled Page - Windows Internet Explorer
http://localhost:4978:
Google
Untitled Page
Page
Local intranet | Protected Mode: On
100%

Request Submitted

Hello, **Scott Hanselman**, we received your notice that you hurt here: **Left arm, Torso**

Figure C-2

Step 1: A Basic ASP.NET Application

Figure C-1 shows our generic Patient Form for our Chiropractic Clinic.

The code for the patient form as seen in Listing C-1 has an empty code-behind class. We've removed some minor markup for the sake of brevity.

You see applications like this every day that are created by composing basic HTML input elements into more and more complicated forms. What can be done with a simple application like this to make it not only a more compelling user experience, but ultimately more *usable?*

Listing C-1: Patient Form

ASPX

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind=
"PatientForm.aspx.cs" Inherits="SLPersonProject.PatientForm" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>ACME Chiropratic Patient Form</title>
</head>
<body>
    <form id="form1" runat="server">
    <h1>ACME Chiropratic Patient Form</h1>
    <div>
        <table border="1">
            <tr>
                <td>First name:</td>
                <td>
                    <asp:TextBox ID="FirstName" runat="server"></asp:TextBox>
                    <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
                        runat="server" ControlToValidate="FirstName"
                        ErrorMessage="*"></asp:RequiredFieldValidator>
                </td>
            </tr>
            <tr>
                <td>Last name:</td>
                <td>
                    <asp:TextBox ID="LastName" runat="server"></asp:TextBox>
                    <asp:RequiredFieldValidator ID="RequiredFieldValidator2"
                        runat="server" ControlToValidate="LastName"
                        ErrorMessage="*"></asp:RequiredFieldValidator>
                </td>
            </tr>
            <tr><td colspan="2">Where does it hurt? </td></tr>
            <tr>
                <td style="width: 125px">
       <asp:CheckBox ID="headCheckBox" runat="server" Text="Head" /><br />
       <asp:CheckBox ID="leftarmCheckBox" runat="server" Text="Left arm" /><br />
```

Continued

```
        <asp:CheckBox ID="rightarmCheckBox" runat="server" Text="Right arm" />
<br />
        <asp:CheckBox ID="torsoCheckBox" runat="server" Text="Torso" /><br />
        <asp:CheckBox ID="leftlegCheckBox" runat="server" Text="Left leg" />
<br />
        <asp:CheckBox ID="rightlegCheckBox" runat="server" Text="Right leg" />
                 </td>
                 <td>What goes here?</td>
             </tr>
             <tr>
                 <td colspan="2" align="right">
                    <asp:Button ID="SubmitButton" runat="server"
                     Text="Submit Doctor Request" PostBackUrl="~/Complete.aspx" />
                 </td>
             </tr>
         </table>
    </div>
    </form>
</body>
</html>
```

Finding Vector-Based Content

A professional ASP.NET programmer will often find themselves adding an ''active element'' to an existing application to make the end user experience richer. Let's add a picture of a human body that users can click on to indicate their areas of pain, but you'll add the constraint that the picture should interact with the existing checkboxes so that server-side code won't need to change.

You may not be an artist, so you can start by looking for some vector- based clip art that you can leverage in the application. Bitmaps are exactly that — maps of bits. When you scale a 100 by 100 image to 400 by 400, each pixel is resampled or resized to be 4x larger, but additional information isn't created. Small pixels simply become larger pixels. Vectors, on the other hand, are represented using mathematical equations and can scale to any size without degradation in quality.

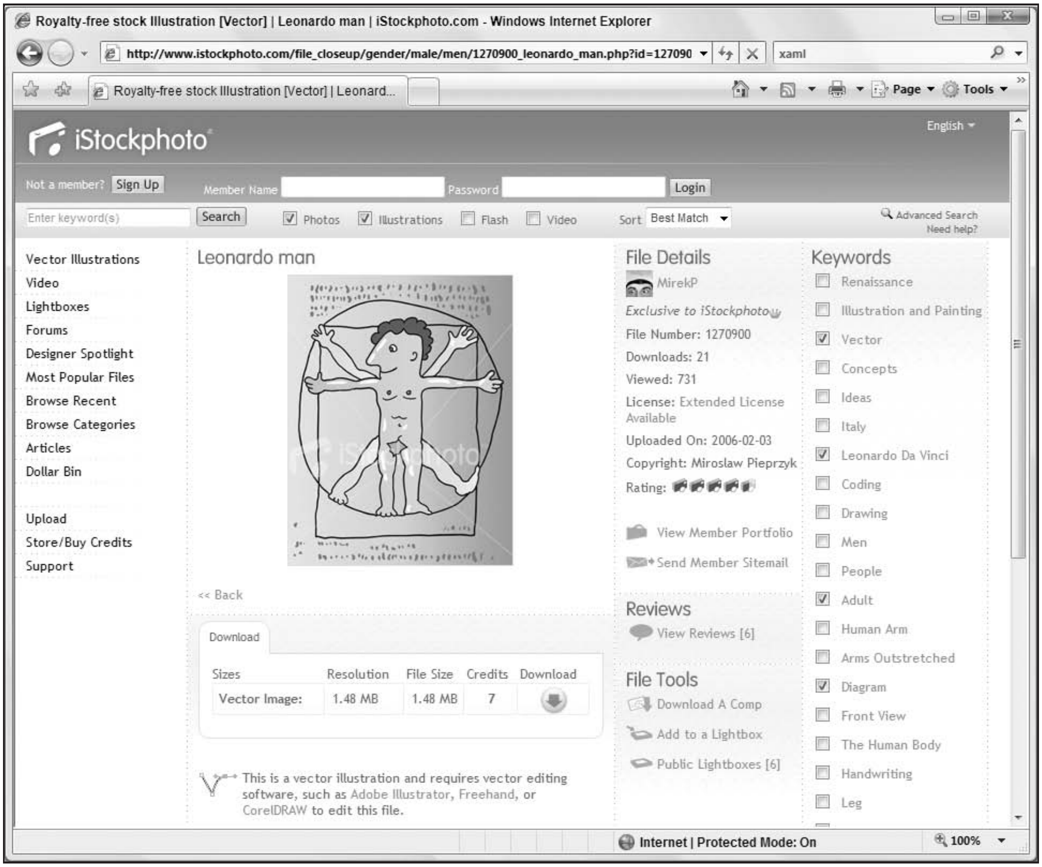
Silverlight uses XAML (eXtensible Application Markup Language) to represent vector graphics. For example, this snippet of XAML draws a gray circle:

```
<Ellipse Width="340" Height="340"  Canvas.Left="50" Canvas.Top="50"
Fill="#FF000000" Opacity="0.3"/>
```

Let's head over to `iStockPhoto.com` and search for ''Adult Vector Diagram'' and download this vector image of a cartoon version of Leonardo da Vinci's *Vitruvian Man* (see Figure C-3). Because it's a vector and not a bitmap it's in EPS (Encapsulated Postscript) format rather than the more familiar PNG, JPG, GIF, or BMP formats that you use every day.

Next, let's download a trial edition of Adobe Illustrator and open the downloaded EPS as shown in Figure C-4.

If you'd like a free alternative to our Cartoon Virtruvian Man, there is a Public Domain alternative at `http://www.hanselman.com/book/asp.net/assets/SilverlightMan.zip`. If you'd rather not use Adobe Illustrator, try the Open Source InkScape at `http://www.inkscape.org/`.


Royalty-free stock Illustration [Vector] | Leonardo man | iStockphoto.com - Windows Internet Explorer
http://www.istockphoto.com/file_closeup/gender/male/men/1270900_leonardo_man.php?id=127090
xaml
Royalty-free stock Illustration [Vector] | Leonard...
Page
Tools
iStockphoto
English
Not a member? Sign Up
Member Name
Password
Login
Enter keyword(s)
Search
Photos
Illustrations
Flash
Video
Sort Best Match
Advanced Search
Need help?
Vector Illustrations
Video
Lightboxes
Forums
Designer Spotlight
Most Popular Files
Browse Recent
Browse Categories
Articles
Dollar Bin
Upload
Store/Buy Credits
Support
Leonardo man
<< Back
Download
File Details
Reviews
View Reviews [6]
File Tools
Download A Comp
Add to a Lightbox
Public Lightboxes [6]
View Member Portfolio
Send Member Sitemail
Keywords
Internet | Protected Mode: On
100%

MirekP
Exclusive to iStockphoto
File Number: 1270900
Downloads: 21
Viewed: 731
License: Extended License Available
Uploaded On: 2006-02-03
Copyright: Miroslaw Pieprzyk
Rating:

Sizes	Resolution	File Size	Credits	Download
Vector Image:	1.48 MB	1.48 MB	7	

This is a vector illustration and requires vector editing software, such as Adobe Illustrator, Freehand, or CorelDRAW to edit this file.

- ☐ Renaissance
- ☐ Illustration and Painting
- ☑ Vector
- ☐ Concepts
- ☐ Ideas
- ☐ Italy
- ☑ Leonardo Da Vinci
- ☐ Coding
- ☐ Drawing
- ☐ Men
- ☐ People
- ☑ Adult
- ☐ Human Arm
- ☐ Arms Outstretched
- ☑ Diagram
- ☐ Front View
- ☐ The Human Body
- ☐ Handwriting
- ☐ Leg

Figure C-3

"Leonardo man" copyright 2006 Miroslaw Pieprzyk, licensed by `iStockphoto.com`.

Zooming in to 300 percent, as shown in Figure C-5, underscores the fact that vectors can be scaled to any resolution.

Converting Vector Content to XAML

We downloaded Mike Swanson's excellent Illustrator to XAML Export plug-in from his site at `www.mikeswanson.com/xamlexport/`. It has some limitations, and they are listed on his site, but it is an excellent tool for bringing vector content into Silverlight.

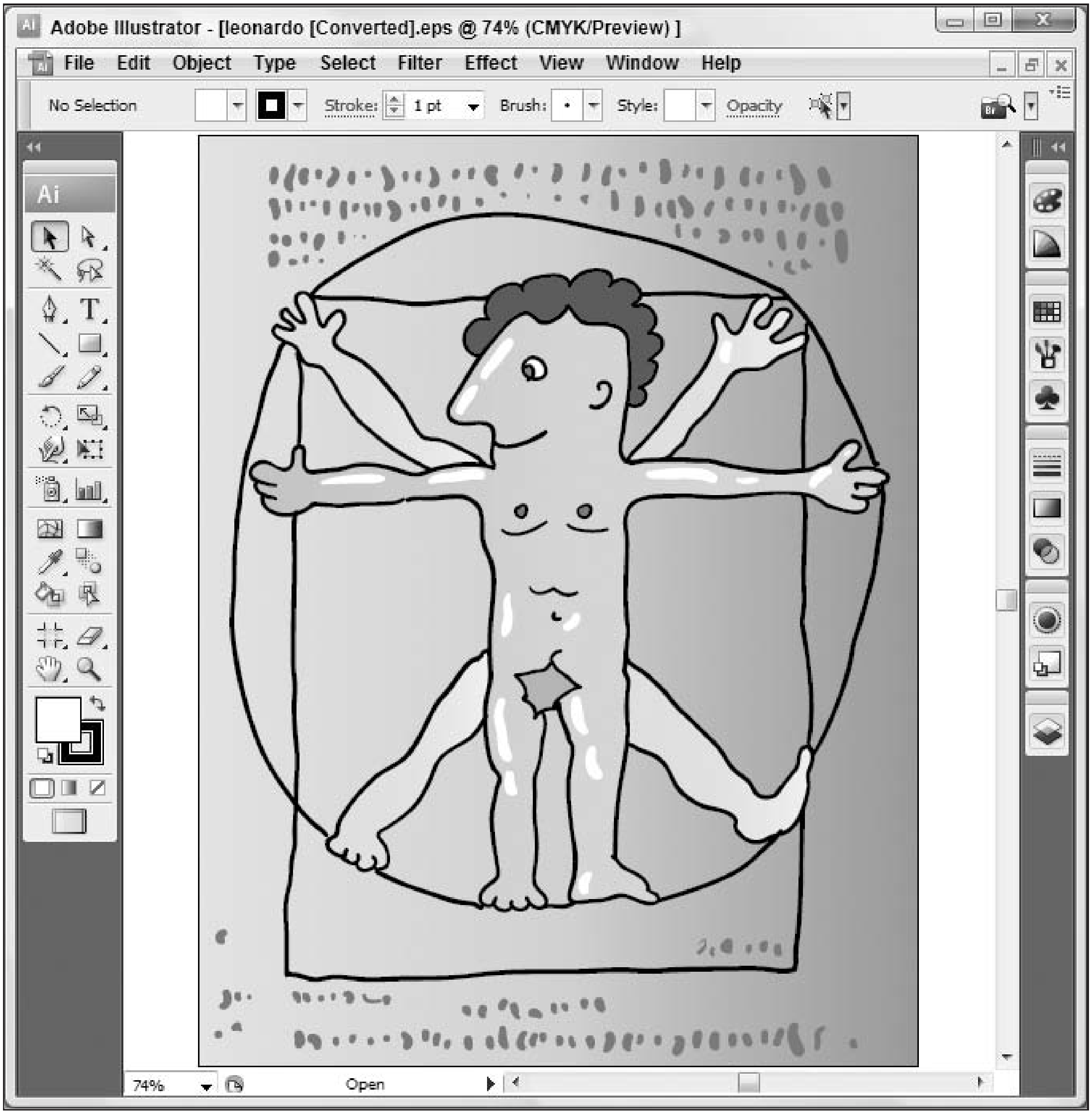
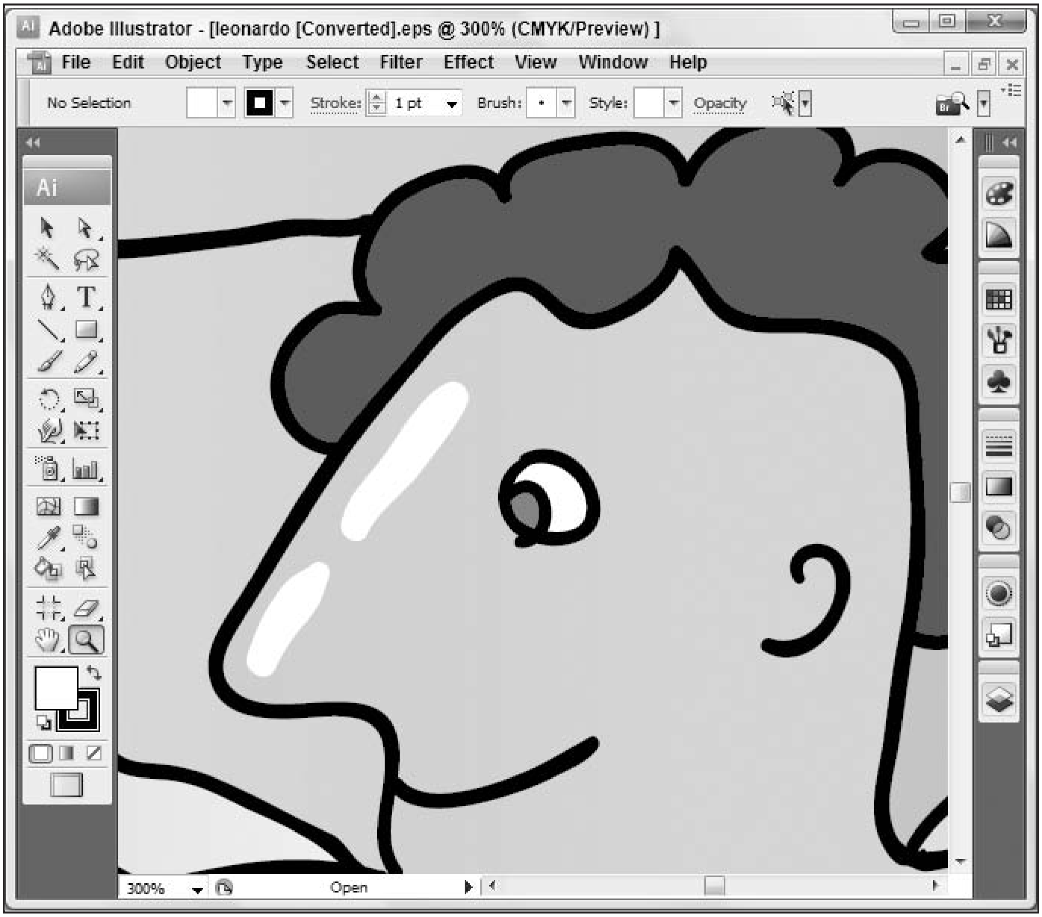

Adobe Illustrator - [leonardo [Converted].eps @ 74% (CMYK/Preview)]
File
Edit
Object
Type
Select
Filter
Effect
View
Window
Help
No Selection
Stroke: 1 pt
Brush:
Style:
Opacity
Ai
74%
Open

Figure C-4

Installation of Mike’s plug-in is simple, just drop the .AIP file into `C:\Program Files\Adobe\Adobe Illustrator CS3\Plug-ins\Illustrator Formats` and you’ll see new options appear in the File ➪ Export dialog box within Illustrator, as shown in Figure C-6.



Adobe Illustrator - [leonardo [Converted].eps @ 300% (CMYK/Preview)]

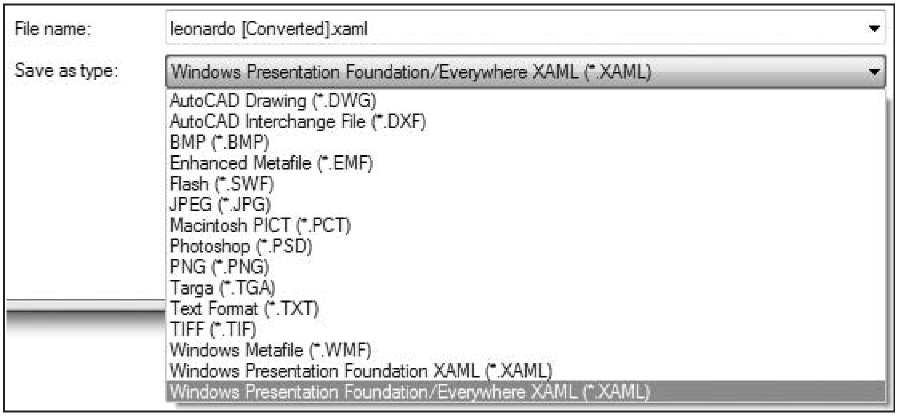
No Selection

Stroke: 1 pt

300%

File
Edit
Object
Type
Select
Filter
Effect
View
Window
Help
Brush:
Style:
Opacity
Ai
Open

Figure C-5



File name: leonardo [Converted].xaml

Save as type: Windows Presentation Foundation/Everywhere XAML (*.XAML)

AutoCAD Drawing (*.DWG)
AutoCAD Interchange File (*.DXF)
BMP (*.BMP)
Enhanced Metafile (*.EMF)
Flash (*.SWF)
JPEG (*.JPG)
Macintosh PICT (*.PCT)
Photoshop (*.PSD)
PNG (*.PNG)
Targa (*.TGA)
Text Format (*.TXT)
TIFF (*.TIF)
Windows Metafile (*.WMF)
Windows Presentation Foundation XAML (*.XAML)
Windows Presentation Foundation/Everywhere XAML (*.XAML)

Figure C-6

There are two options, one for WPF XAML usable in a .NET 3.0 Windows Application and one using the old code-name for Silverlight, Windows Presentation Foundation/Everywhere or WPF/E. You'll select the last one as you're using Silverlight. The only difference in the documents is the root node of the XML.

The resulting XAML document is about 256 KB and starts out like this:

```
<!-- Generated by Adobe Illustrator CS -> XAML Export Plug-In Version 0.16      -->
<!-- For questions, contact Mike Swanson: http://www.mikeswanson.com/XAMLExport -->

<Canvas Width="612.000000" Height="792.000000"
  xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Canvas>
  <!-- Layer 1/<Path> -->
  <Path Data="F1 M 612.000000,792.000000 L 0.000000,792.000000 L
0.000000,0.000000 L 612.000000,0.000000 L 612.000000,792.000000 Z">
...the other 255k removed...
```

How do you know if the export succeeded and our little man looks correct? There are a few ways.

Tools for Viewing and Editing XAML

There are a number of ways for you to view or edit our newly created XAML file. You can use Internet Explorer, Visual Studio 2008's XAML Editor, or Microsoft Expression Blend.

Internet Explorer with a Minimal Silverlight Page

First, you create a minimal static HTML page to view your XAML, as shown in Figure C-7.

The code is fairly simple. First, the `default.html` file shown in Listing C-2a includes a `div` to hold the Silverlight control.

Listing C-2a: Helloworld.html

HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3c.org/TR/1999/REC-html401-19991224/loose.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Hello Dude</title>
  <script type="text/javascript" src="Silverlight.js"></script>
  <script type="text/javascript" src="HelloWorld.js"></script>
  <script type="text/javascript" src="Dude.xaml.js"></script>
  <style type="text/css">
        .silverlightHost {
               height: 792px;
               width: 612px;
        }
  </style>
</head>
<body>
```

```
            <div id="SilverlightControlHost" class="silverlightHost">
             <script type="text/javascript">
                   createSilverlight();
            </script>
     </div>
   </body>
   </html>
```

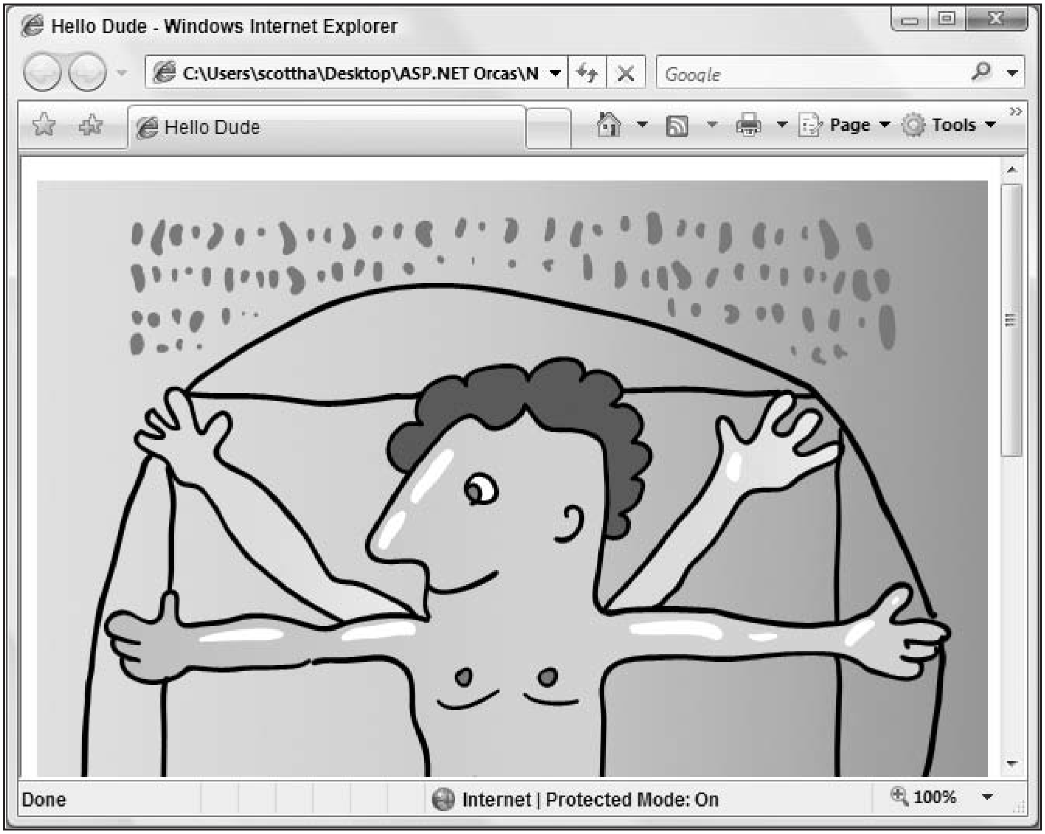

Hello Dude - Windows Internet Explorer
C:\Users\scottha\Desktop\ASP.NET Orcas\N
Google
Hello Dude
Page
Tools
Done
Internet | Protected Mode: On
100%

Figure C-7

The `div` with the ID `SilverlightControlHost` calls a method called `createSilverlight` that lives in HelloWorld.js, shown in Listing C-2b. We've separated the JavaScript files for the sake of tidiness, but you're welcome to lay the files out however you like.

Listing C-2b: Helloworld.js

JavaScript

```
function createSilverlight()
{
  var scene = new HelloWorld.Page();
```

Continued

```
    Silverlight.createObjectEx({
            source: "Dude.xaml",
            parentElement: document.getElementById("SilverlightControlHost"),
            id: "SilverlightControl",
            properties: {
                   width: "100%",
                   height: "100%",
                   version: "1.0"
            },
            events: {
                   onLoad: Silverlight.createDelegate(scene, scene.handleLoad)
            }
     });
   }


   if (!window.Silverlight)
     window.Silverlight = {};

   Silverlight.createDelegate = function(instance, method) {
     return function() {
            return method.apply(instance, arguments);
     }
   }
```

Notice the three lines called out in Listing C-2b. The ID of the `div` from our HTML file is referenced in the call to `getElementById` above. Our XAML file is referenced in the `source:` property.

You can hook up JavaScript events to Silverlight, as shown in Listing C-2c. A good naming convention is to put these events in a file named `yourxamlfile.xaml.js`, so I've named mine `Dude.xaml.js`.

Listing C-2c: Dude.xaml.js

JavaScript

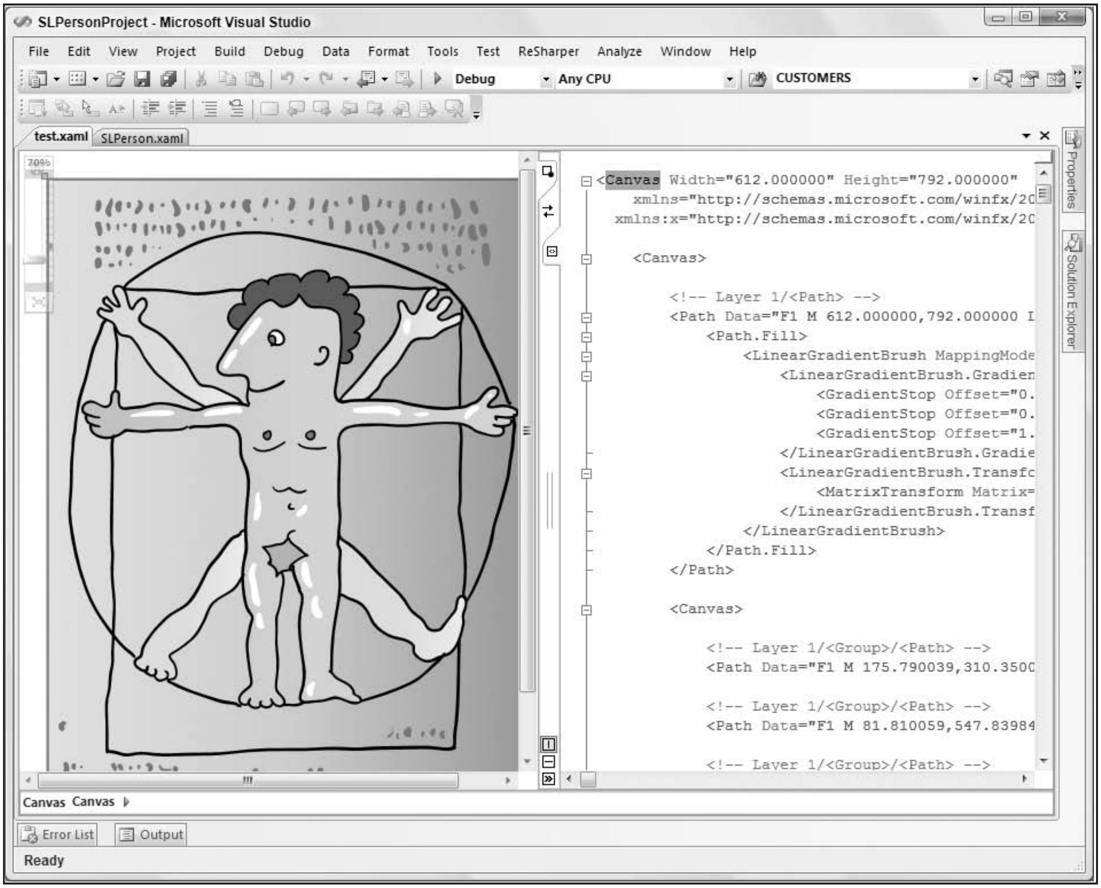
```
if (!window.HelloWorld)
 window.HelloWorld = {};

HelloWorld.Page = function() {}

HelloWorld.Page.prototype = {
 handleLoad: function(control, userContext, rootElement)
 {
         this.control = control;
         // Sample event hookup:
         rootElement.addEventListener("MouseLeftButtonDown",
            Silverlight.createDelegate(this, this.handleMouseDown));
 },
// Sample event handler
 handleMouseDown: function(sender, eventArgs) {
         // The following line of code shows how to find an
```

Continued

```
            // element by name and call a method on it.
            // this.control.content.findName("something").Begin();
    }
   }
```

Visual Studio 2008's Built-in XAML Editor

Visual Studio 2008 includes a XAML editor for editing WPF XAML rather than Silverlight. However, it'll let you move things around and confirm our layout if you change the `http://schemas.microsoft.com/winfx/2007` namespace to `http://schemas.microsoft.com/winfx/2006/xaml/presentation` in our XAML file. This isn't recommended, but it's an interesting exercise in exploring the subtle differences between WPF and Silverlight's implementation of XAML. Figure C-8 shows our XAML file edited in Visual Studio in a split-screen.


SLPersonProject - Microsoft Visual Studio
File
Edit
View
Project
Build
Debug
Data
Format
Tools
Test
ReSharper
Analyze
Window
Help
Debug
Any CPU
CUSTOMERS
test.xaml
SLPerson.xaml
70%
Properties
Solution Explorer
Canvas
Canvas
Error List
Output
Ready

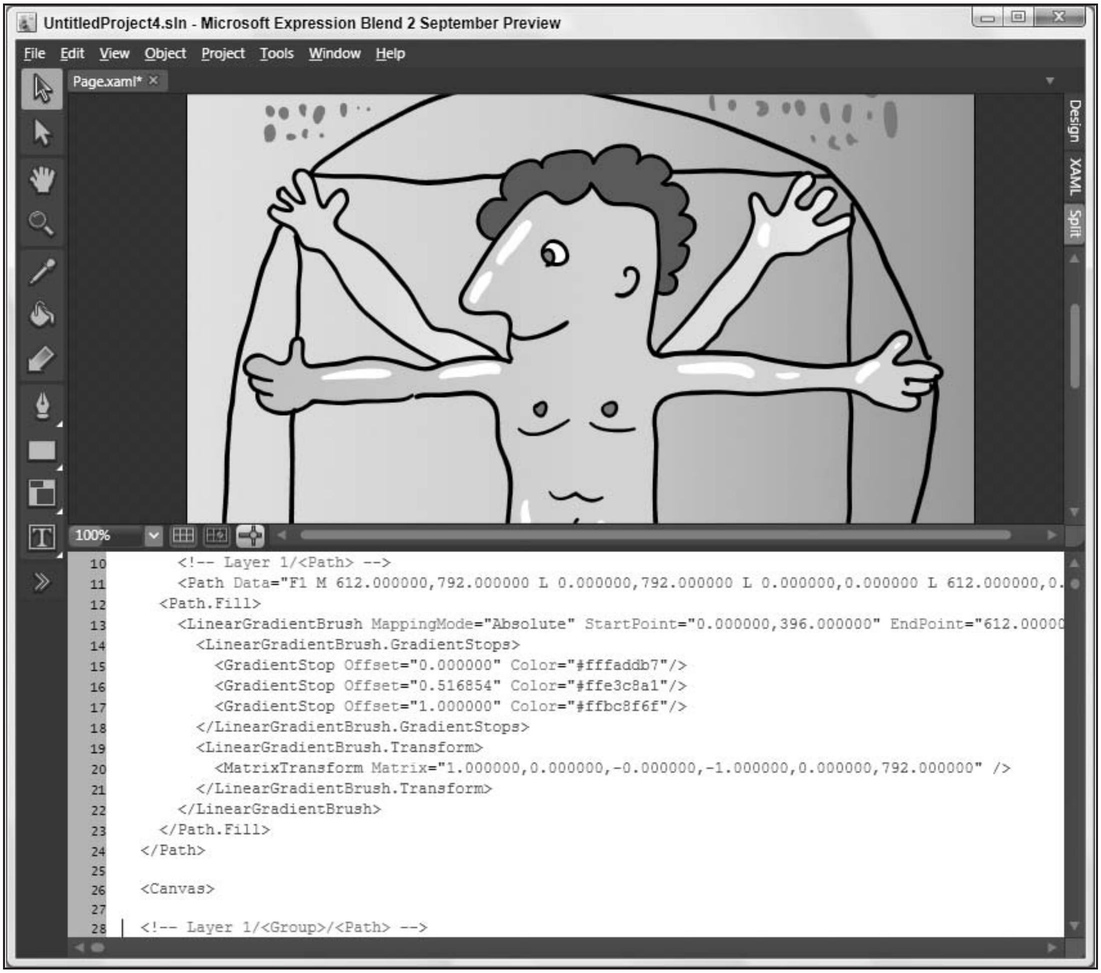
```
<Canvas Width="612.000000" Height="792.000000"
  xmlns="http://schemas.microsoft.com/winfx/20
 xmlns:x="http://schemas.microsoft.com/winfx/20

  <Canvas>

    <!-- Layer 1/<Path> -->
    <Path Data="F1 M 612.000000,792.000000 I
      <Path.Fill>
        <LinearGradientBrush MappingMode
          <LinearGradientBrush.Gradien
            <GradientStop Offset="0.
            <GradientStop Offset="0.
            <GradientStop Offset="1.
          </LinearGradientBrush.Gradie
          <LinearGradientBrush.Transfc
            <MatrixTransform Matrix=
          </LinearGradientBrush.Transf
        </LinearGradientBrush>
      </Path.Fill>
    </Path>

    <Canvas>

      <!-- Layer 1/<Group>/<Path> -->
      <Path Data="F1 M 175.790039,310.3500

      <!-- Layer 1/<Group>/<Path> -->
      <Path Data="F1 M 81.810059,547.83984

      <!-- Layer 1/<Group>/<Path> -->
```

Figure C-8

Microsoft Expression Blend

At the time of these writing, Expression Blend 2 was in Beta and not yet released. Blend is a designer focused editor for XAML files, animations, and Silverlight projects. Blend shares the same project file format as Visual Studio, so you can open CSPROJ files and SLN files and move seamlessly between Blend and VS.

Figure C-9 shows our XAML file being edited in Expression Blend. Blend gives you complete control over XAML files, and you'll use it to extend the image for your application. For the application, you'll add red dots over each of the body parts represented in the original applications by checkboxes. You'll name them so they are accessible from JavaScript and hook up events to both the checkboxes and within Silverlight so they both stay in sync.


UntitledProject4.sln - Microsoft Expression Blend 2 September Preview
File
Edit
View
Object
Project
Tools
Window
Help
Page.xaml*
Design
XAML
Split
100%

```
    <!-- Layer 1/<Path> -->
    <Path Data="F1 M 612.000000,792.000000 L 0.000000,792.000000 L 0.000000,0.000000 L 612.000000,0.
  <Path.Fill>
    <LinearGradientBrush MappingMode="Absolute" StartPoint="0.000000,396.000000" EndPoint="612.00000
      <LinearGradientBrush.GradientStops>
        <GradientStop Offset="0.000000" Color="#fffaddb7"/>
        <GradientStop Offset="0.516854" Color="#ffe3c8a1"/>
        <GradientStop Offset="1.000000" Color="#ffbc8f6f"/>
      </LinearGradientBrush.GradientStops>
      <LinearGradientBrush.Transform>
        <MatrixTransform Matrix="1.000000,0.000000,-0.000000,-1.000000,0.000000,792.000000" />
      </LinearGradientBrush.Transform>
    </LinearGradientBrush>
  </Path.Fill>
</Path>

<Canvas>

<!-- Layer 1/<Group>/<Path> -->
```

Figure C-9

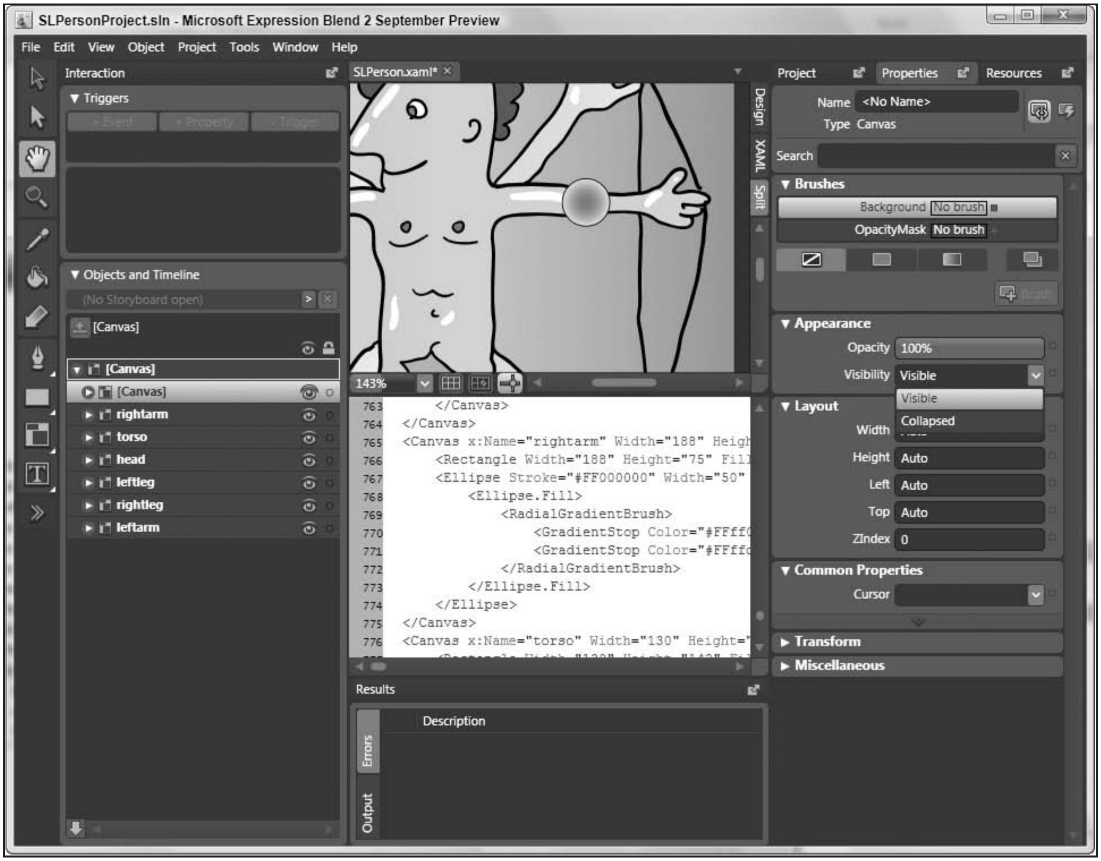
Adding Active Elements with Blend

Open up your exported XAML file in Blend and perform the following tasks:

1. Using the basic drawing tools in the toolbox, draw a circle over the left and right arms, left and right legs, the head, and the torso. Feel free to color them any way you like. I've used a gradient.
2. Name each of the newly drawn circles leftLeg, rightArm, and so on, by selecting them and typing in a name in the Properties pane.

You'll notice while Blend is all black and has a different UI than you're used to, the metaphors of object selection, properties panes, and naming are familiar from your experience using Visual Studio. Blend is almost like ''Visual Studio for Designers.''

Alternatively, if you don't like using the visual editor in Blend as seen in Figure C-10, you can just copy and paste the XAML directly.


SLPersonProject.sln - Microsoft Expression Blend 2 September Preview
File Edit View Object Project Tools Window Help
Interaction
Triggers
+ Event
+ Property
- Trigger
Objects and Timeline
(No Storyboard open)
[Canvas]
[Canvas]
[Canvas]
rightarm
torso
head
leftleg
rightleg
leftarm
SLPerson.xaml*
Design
XAML
Split
143%
Results
Description
Errors
Output
Project
Properties
Resources
Search
Brushes
Appearance
Layout
Common Properties
Transform
Miscellaneous
Visible
Collapsed

Name <No Name>
Type Canvas
Background No brush
OpacityMask No brush
Opacity 100%
Visibility Visible
Height Auto
Left Auto
Top Auto
ZIndex 0
Cursor

```
        </Canvas>
    </Canvas>
    <Canvas x:Name="rightarm" Width="188" Heigh
        <Rectangle Width="188" Height="75" Fil
        <Ellipse Stroke="#FF000000" Width="50"
            <Ellipse.Fill>
                <RadialGradientBrush>
                    <GradientStop Color="#FFff
                    <GradientStop Color="#FFff
                </RadialGradientBrush>
            </Ellipse.Fill>
        </Ellipse>
    </Canvas>
    <Canvas x:Name="torso" Width="130" Height="
```

Figure C-10

Editing in Notepad

You can edit in your favorite XML editor or even Notepad. Each ‘‘pain dot’’ is a snippet of XAML that resembles Listing C-3:

Listing C-3: A XAML Ellipse

```
<Canvas x:Name="leftarm" Width="193" Height="60" Canvas.Left="377.762"
Canvas.Top="261.857">
    <Rectangle Width="193" Height="60" Fill="Transparent"></Rectangle>
    <Ellipse Stroke="#FF000000" Width="50" Height="50"
          Visibility="Collapsed"
         Canvas.Left="57" Canvas.Top="8">
        <Ellipse.Fill>
            <RadialGradientBrush>
               <GradientStop Color="#FFff0000" Offset="0"/>
               <GradientStop Color="#FFffddaa" Offset="1"/>
            </RadialGradientBrush>
        </Ellipse.Fill>
    </Ellipse>
</Canvas>
```

Listing C-3 is an example of an ellipse with a filled gradient between red and a shade of light-gray. The only differences between the six pain dots are these:

- `x:Name` attribute: In order for a Silverlight element to be ‘‘addressable’’ from JavaScript, it should have a name.
- `Canvas.Left="x" Canvas.Top="y"` attributes: These indicate the position of the elements within the larger canvas.

You can certainly edit XAML manually in Notepad and preview your changes in Internet Explorer via the very repetitive ‘‘Edit, Refresh in Browser, Edit, Refresh in Browser,’’ technique if you like, but we recommend using Blend.

> **Notice also the Visibility attribute. You want these dots to be initially invisible, so they are marked as `Visibility="Collapsed"`. You will toggle the dot’s visibility in JavaScript code. Draw all the dots and position them first before you make their `Visibility="Collapsed"`. Just before you hit the dots, you’ll have a person that resembles Figure C-11.**

Integrating with Your Existing ASP.NET Site

All right, now that you have our imported and edited ‘‘Chiro Dude,’’ let’s add him to the site. You have a few choices. You can copy over the Silverlight.js and the other assets and simply add them to your Visual Studio Project as content. The Javascript files can be added as standard script references on your ASPX pages and the Silverlight `div` added manually.

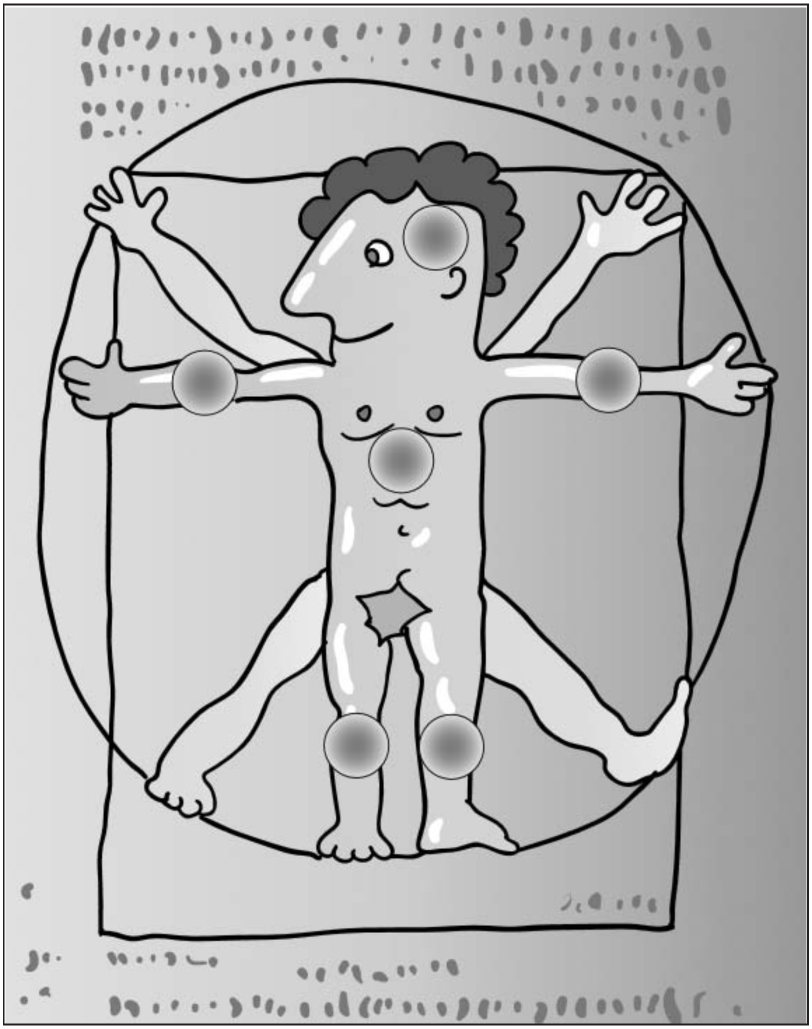


Figure C-11

However, Microsoft often has ''Futures Releases'' or ''Previews'' that showcase coming technologies that make things easier. The ASP.NET 3.5 Extensions Preview at the time of this writing is available at `http://www.asp.net/downloads/3.5-extensions/`. It includes a preview of some new controls that will arrive around the first half of 2008 as a downloadable add-on to the .NET Framework 3.5. These controls will simply be added into the Visual Studio Toolbox and make things easier. You should be able to get these controls within a few months of the publication of this very book!

Remember earlier when we created three JavaScript files and needed to check ID attributes to instantiate a new Silverlight Control? The `asp:Silverlight`and improved `asp:ScriptManager`controls will likely make that process easier.

An update of Listing C-1 using these controls looks like Listing C-4, with some repetition removed for brevity. Pay special attention to the `ScriptReferences` in the `ScriptManager`. One uses `Name=` and one `Path=`. That is not a typo. The Silverlight JavaScript is referred to by name and is served from an Assembly Resource. It's compiled into a DLL and served from there by `ScriptResource.axd`, an HttpHandler for doing just that. The Dude.xaml.js will be served up directly from disk like any other JavaScript file, except you didn't have to write the `script` tag because it's handled for you.

Listing C-4: Markup using ASP.NET 3.5 Extensions

ASPX

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
     <asp:ScriptManager ID="ScriptManager1" runat="server">
         <Scripts>
             <asp:ScriptReference Name="SilverlightControl.js" />
             <asp:ScriptReference Path="Dude.xaml.js" />
         </Scripts>
     </asp:ScriptManager>
     <h1>Patient Form</h1>
     <div>
         <table border="1">
 ...OMMITED...
             <tr>
                 <td colspan="2">Where does it hurt? </td>
             </tr>
             <tr>
                 <td style="width: 125px">
     <asp:CheckBox ID="headCheckBox" runat="server" Text="Head" /><br />
     <asp:CheckBox ID="leftarmCheckBox" runat="server" Text="Left arm" /><br />
     <asp:CheckBox ID="rightarmCheckBox" runat="server" Text="Right arm" /><br />
     <asp:CheckBox ID="torsoCheckBox" runat="server" Text="Torso" /><br />
     <asp:CheckBox ID="leftlegCheckBox" runat="server" Text="Left leg" /><br />
     <asp:CheckBox ID="rightlegCheckBox" runat="server" Text="Right leg" />
                 </td>
                 <td>
                     <asp:Silverlight runat="server" ID="Xaml1" Height="475"
                            Width="367" Source="Dude.xaml"
                         ClientType="Custom.SLPerson"></asp:Silverlight>
                 </td>
             </tr>
             <tr>
                 <td colspan="2" align="right">
                       <asp:Button ID="SubmitButton" runat="server" Text="Submit
   Doctor Request" PostBackUrl="~/Complete.aspx" />
                 </td>
             </tr>
         </table>
    </div>
    </form>
</body>
</html>
```

In order to get Listing C-4 to work, you'll need the ASP.NET 3.5 Extensions from `http://www.asp.net/downloads/3.5-extensions/`. The release date of this new package was not available at the time of this writing, but watch `www.asp.net` for all the latest details.

At this point, you've added the control but there's no interactivity. He's on the page, but he doesn't do anything. You need to expose the named dots that you created earlier. You will use JavaScript to find them within the XAML document and assign them to JavaScript variables. Silverlight is very good at being transparent. Once you've gotten hold of a Silverlight object within JavaScript, you can treat it just like any other JavaScript object. Notice that the ID in this case of the Silverlight object is ''Xaml1.''

Receiving Silverlight Events in JavaScript

You start your JavaScript by creating a ''class'' called `personElements` with each of the dots. They are initially set to null because you find them in the `initializeComponent` method that acts as your constructor. Notice how `slhost`is passed into `initializeComponent`. Then you access its `content` node and use `findName` to grab each of the named body part dots, storing them away in variables as seen in Listing C-5.

Listing C-5: Dude.xaml.js

JavaScript

```
/// <reference name="MicrosoftAjax.js"/>
/// <reference name="SilverlightControl.js"/>

personElements = function() {
    this.leftarm = null;
    this.rightarm = null;
    this.head = null;
    this.leftleg = null;
    this.rightleg = null;
    this.torso = null;
}
personElements.prototype = {
    initializeComponent: function(slhost) {
        var host = slhost.content;

        this.leftarm = host.findName("leftarm");
        this.rightarm = host.findName("rightarm");
        this.head = host.findName("head");
```

Continued

```
        this.leftleg = host.findName("leftleg");
        this.rightleg = host.findName("rightleg");
        this.torso = host.findName("torso");

    }
}

Type.registerNamespace("Custom");

Custom.SLPerson = function(element) {
    Custom.SLPerson.initializeBase(this, [element]);
    this._designer = new personElements();
}

Custom.SLPerson.prototype = {
    initialize : function() {
        Custom.SLPerson.callBaseMethod(this, 'initialize');

        // Call on the component initialized to get the
        // specific component's XAML element fields.
        this._designer.initializeComponent(this.get_element());

        // Hookup event handlers as required in this custom type.
        var onClick = Function.createDelegate(this, this._onPersonClick);

    this.addEventListener(this._designer.leftarm, "mouseLeftButtonUp", onClick);
    this.addEventListener(this._designer.rightarm, "mouseLeftButtonUp", onClick);
    this.addEventListener(this._designer.head, "mouseLeftButtonUp", onClick);
    this.addEventListener(this._designer.leftleg, "mouseLeftButtonUp", onClick);
    this.addEventListener(this._designer.rightleg, "mouseLeftButtonUp", onClick);
    this.addEventListener(this._designer.torso, "mouseLeftButtonUp", onClick);
    },
    _onPersonClick : function(sender, e) {
        var region = sender.Name;
        this.toggleVisibility(region);

        var elem = $get(region + "CheckBox");
        elem.checked = !elem.checked;
    },

    toggleVisibility : function(region) {
        var elem = this._designer[region];
        var wasVisible = (elem.children.getItem(1).Visibility == "Visible");
        elem.children.getItem(1).Visibility = wasVisible ? "Collapsed" : "Visible";
    }
}
Custom.SLPerson.registerClass('Custom.SLPerson', Sys.UI.Silverlight.Control);
```

After finding those parts, you hook up to the `mouseLeftButtonUp` event and attach it to `_onPersonClick`. That method will get called each time you click a dot.

Within `_onPersonClick` you call a new method `toggleVisibility` that will not only change the visibility of the dot, but also use an ASP.NET Ajax JavaScript method `$get()` to grab onto a similarly named HTML checkbox and toggle its `checked` property.

At this point, clicking on the Chiro Dude in certain spots toggles the dots and checkboxes simultaneously. However, you also need to hook event handlers up to the checkboxes themselves so that they toggle the dots within the Chiro Dude.

Accessing Silverlight Elements from JavaScript Events

You can use the `$find()` method to access our Silverlight control. What's this? There's `$find()` and `$get()`? What's the difference between these two methods?

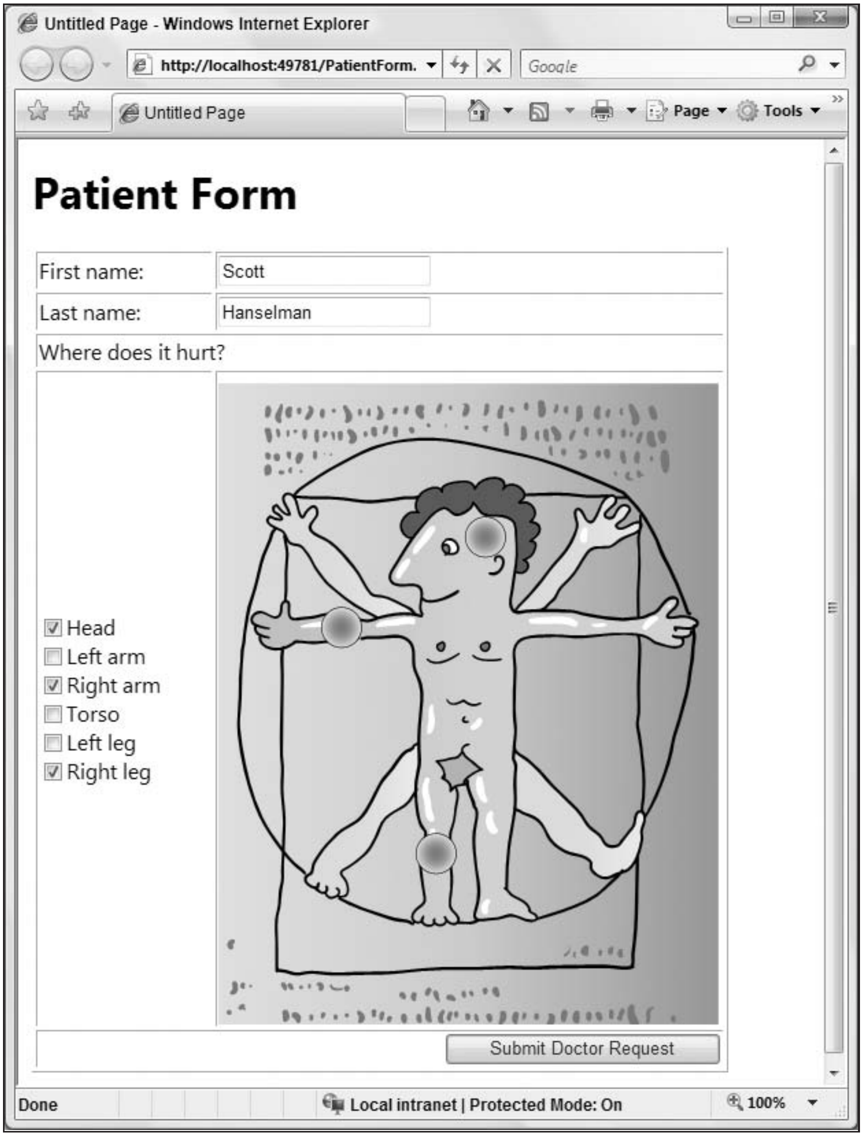
Well, `$find()` is a shortcut for `Sys.Application.findComponent`, whereas `$get()` is an alias for the `getElementById` method. The `$get()` method will work on any browser, even those without support for `getElementById`. Generally, `$find()` is useful for getting a reference to a Silverlight control from within JavaScript, whereas `$get()` is typically used for getting references to standard HTML controls, DIVS, and SPANS.

Now, let's add these attributes to our check boxes:

```
<asp:CheckBox ID="headCheckBox" runat="server" Text="Head"
OnClick="$find('Xaml1').toggleVisibility('head')" />
<br />
<asp:CheckBox ID="leftarmCheckBox" runat="server" Text="Left arm"
OnClick="$find('Xaml1').toggleVisibility('leftarm')" />
<br />
<asp:CheckBox ID="rightarmCheckBox" runat="server" Text="Right arm"
OnClick="$find('Xaml1').toggleVisibility('rightarm')" />
<br />
<asp:CheckBox ID="torsoCheckBox" runat="server" Text="Torso"
 OnClick="$find('Xaml1').toggleVisibility('torso')" />
<br />
<asp:CheckBox ID="leftlegCheckBox" runat="server" Text="Left leg"
OnClick="$find('Xaml1').toggleVisibility('leftleg')" />
<br />
<asp:CheckBox ID="rightlegCheckBox" runat="server" Text="Right leg"
OnClick="$find('Xaml1').toggleVisibility('rightleg')" />
```

Here you've added a `$find()` call to grab your Silverlight controls. Then you call the `toggleVisbility` method that you added in Listing C-5. You've completed the cycle and now users can click the check boxes or the Chiro Dude in order to make their selection. And, because the resulting Form POST still uses the values of the check boxes, you haven't had to change any server-side code.

The completed application is shown in Figure C-12. Notice that the check boxes and Chiro Dude's pain dots are in sync.


Untitled Page - Windows Internet Explorer
http://localhost:49781/PatientForm.
Google
Untitled Page
Page
Tools
Done
Local intranet | Protected Mode: On
100%

Patient Form

First name: Scott

Last name: Hanselman

Where does it hurt?

☑ Head
☐ Left arm
☑ Right arm
☐ Torso
☐ Left leg
☑ Right leg

Submit Doctor Request

Figure C-12

Summary

Silverlight 1.0 has a very clean, very natural programming model that feels familiar to developers who have programmed against the JavaScript DOM or worked with AJAX code of any kind. The bridge between Silverlight and JavaScript is seamless, allowing ASP.NET developers to mix and match Silverlight and HTML as they like.

Programming with Silverlight will get even easier in the near future when a minor release adds Silverlight Server Controls and an improved ScriptManager to ASP.NET 3.5.

D

ASP.NET Online Resources

Author Blogs

Bill Evjen: `www.geekswithblogs.net/evjen`

Scott Hanselman: `www.hanselman.com/blog/`

Devin Rader: `www.geekswithblogs.net/devin`

ASP.NET Influential Blogs

Kent Sharkey: `www.acmebinary.com/blog`

Rob Howard: `weblogs.asp.net/rhoward/`

Scott Guthrie: `weblogs.asp.net/scottgu`

Steve Smith: `blogs.aspadvice.com/ssmith/`

G. Andrew Duthie: `blogs.msdn.com/gduthie/`

Scott Mitchell: `scottonwriting.net/sowBlog/`

Scott Watermasysk: `scottwater.com/blog/`

Nikhil Kothari: `www.nikhilk.net/`

Dave Sussman: `www.daveandal.net/daveroom/diary.asp`

Mike Pope: `www.mikepope.com/blog/`

Web Sites

123ASPX Directory: `www.123aspx.com`

4 Guys from Rolla: `www.4guysfromrolla.com`

Angry Coder: `www.angrycoder.com`

ASP 101: `www.asp101.com`

ASP Alliance: `www.aspalliance.com`

ASP Alliance Lists: `www.aspadvice.com`

The ASP.NET Developer Portal: `msdn2.microsoft.com/asp.net`

ASP.NET Homepage: `www.asp.net`

ASP.NET Resources: `www.aspnetresources.com`

ASP.NET World: `www.aspnetworld.com`

DotNetJunkies: `www.dotnetjunkies.com`

GotDotNet: `www.gotdotnet.com`

International .NET Association: `www.ineta.org`

Microsoft's ASP.NET AJAX Site: `www.asp.net/ajax/`

Microsoft's Classic ASP Site: `msdn2.microsoft.com/en-us/library/aa286483.aspx`

Microsoft Developer Centers: `msdn2.microsoft.com/developercenters`

Microsoft Forums: `forums.microsoft.com`

Microsoft Newsgroups: `msdn.microsoft.com/newsgroups/`

`Microsoft's Open Source Project Community: www.codeplex.com`

.NET 247: `www.dotnet247.com`

RegExLib: `www.regexlib.com`

The ServerSide .NET: `www.theserverside.net`

XML for ASP.NET: `www.xmlforasp.net`

Index

SYMBOLS

- **$ (dollar sign), currency translation,** 1393–1394
- *** (asterisk), URL Authorization,** 1431
- **“. . . <>”, calling descendants,** 508–509
- **“<>”, calling elements,** 508–509
- **?, URL Authorization,** 1431
- `! important` **attribute,** 883–884

A

- **<a> element,** 273–274
- **Absolute positioning, CSS elements,** 882
- **accelerator keys,** 109
- **Access Control Lists.** ***See*** **ACLs (Access Control Lists)**
- **access rules,** 1502–1507, 1510
- **AccessDataSource control,** 307
- `AccessKey` **attribute, hot-keys,** 110
- **Accordion control, AJAX,** 986–988
- **AccordionPane controls,** 986–987
- **ACLs (Access Control Lists)**
  - adding rules to, 1163–1164
  - overview of, 1160
  - querying information about, 1161–1163
  - removing rules from, 1164–1166
- `AcquireRequestState`**,** `Session` **object,** 1036–1037
- **acronyms, design guidelines,** 498
- **Actions pane, IIS Manager,** 570
- `Activate` **event, Wizard server control,** 185–186
- **Active Directory Application Mode (ADAM),** 600–602
- **Active Server Pages (ASP),** 63
- `ActiveDirectory-MembershipProvider`**,** 600–602
- **ActiveX component, installing private assembiles,** 1312–1313
- **ActiveX DLL**
  - adding to References section of project, 1303
  - using COM objects in ASP.NET, 1304–1305, 1307–1308
- **ADAM (Active Directory Application Mode),** 600–602
- **adapters,** 893
- **<add/> element,** 1483–1484, 1486
- **Add Application Extensions Mapping dialog, IIS,** 5, 1287–1288
- **Add Connection dialog box,** 419–420
- **Add Field dialog, GridView,** 330–331
- **Add Reference dialog,** 1301, 1303, 1307
- **Add Service Reference dialog,** 1370–1371, 1377
- **Add Web Reference dialog,** 1336–1338
- **<**`add`**> element**
  - `<eventMappings>` configuration, 1481
  - adding SQL Server data provider, 746
  - application-specific settings, 1440
  - SQL Server cache invalidation, 1093
  - working with personalization properties, 726, 731–732, 735
- `AddAccessRule( )` **method, ACLs,** 1164
- `AddAttribute( )` **method, server controls,** 1216
- `AddAttributesToRender( )` **method, server controls,** 1217–1220
- `AddRemoveProgramsIcon` **property, Windows Installer,** 1553
- `AddStyleAttribute`**, server controls,** 1220
- `AddUsersToRole( )` **method,** `Roles` **class,** 655–656, 659–660, 802–803
- **Adjacent Selectors, CSS,** 868
- **administration**
  - security trimming, 715–720
  - setting up `AdminOnly.aspx` page, 717
  - using Web Site Administration Tool. *See* Web Site Administration Tool
- **Administration Tool**
  - building browser-based, 1470–1476
  - viewing performance counters through, 1468–1470
- **ADO.NET,** 378–454
  - asynchronous command execution. *See* asynchronous command execution
  - asynchronous connections, 454
  - DataList server control, 403–410
  - `DataSet` and `DataTable`, 395–400
  - deleting data, 382
  - inserting data, 380–381
  - ListView server control, 410–419
  - namespaces and classes, 383–384
  - selecting data, 378–380
  - updating data, 381–382

ADO.NET *(continued)*
  using `Command` object, 386–387
  using `Connection` object, 384–386
  using `DataAdapter` object, 389–392
  using `DataReader` object, 387–389
  using Oracle database with ASP.NET 3.5, 400–403
  using parameters, 392–395
ADO.NET, using Visual Studio, 419–432
  creating connection to data source, 419–422
  using `CustomerOrders` DataSet, 427–432
  working with DataSet Designer, 422–427
AdRotator server control, 151–152, 366
Advanced Settings dialog, IIS Manager, 572–574
`after` **element, CSS,** 871
`AggregateCacheDependency` **class,** 1082
AJAX (Asynchronous JavaScript and XML), 895–928
  applications, 902–904
  Aptana open source libraries for, 1588
  building ASP.NET page with, 906–911
  building ASP.NET page without, 904–906
  development of, 896–899
  need for, 895–896
  overview of, 895
  Visual Studio 2008 and, 899–902
AJAX (Asynchronous JavaScript and XML), server-side controls, 911–928
  overview of, 911–912
  ScriptManager control, 912–914
  ScriptManagerProxy control, 914–916
  Timer control, 916–917
  UpdatePanel control, 917–922
  UpdateProgress control, 922–925
  using multiple UpdatePanel controls, 925–928
AJAX Control Toolkit, 929–994
  adding controls to, 934–936
  adding controls to VS2008 Toolbox, 932–934
  downloading and installing, 929–930
  new Visual Studio templates, 931–932
  overview of, 1597
AJAX Control Toolkit control extenders
  AlwaysVisibleControlExtender control, 937–939
  AnimationExtender server control, 939–941
  AutoCompleteExtender control, 941–944
  CalendarExtender control, 944–946
  CollapsiblePanelExtender server control, 946–947
  ConfirmButtonExtender control, 947–948
  DragPanelExtender control, 950–951
  DropDownExtender control, 951–953
  DropShadowExtender control, 953–955
  DynamicPopulateExtender control, 956–959
  FilteredTextBoxExtender control, 959–961
  HoverMenuExtender control, 961–962
  ListSearchExtender control, 962–964
  MaskedEditExtender and MaskedEditValidator controls, 964–966
  ModalPopUpExtender control, 948–949
  MutuallyExclusiveCheckBoxExtender control, 967–968
  NumericUpDownExtender control, 968–969
  overview of, 937
  PagingBulletedListExtender control, 969–970
  PopUpControlExtender control, 970–972
  ResizableControlExtender control, 972–974
  RoundedCornersExtender control, 975–976
  SliderExtender control, 976–977
  SlideShowExtender control, 977–979
  TextBoxWatermarkExtender control, 979–981
  ToggleButtonExtender control, 982–983
  UpdatePanelAnimationExtender control, 983–984
  ValidatorCalloutExtender control, 984–985
AJAX Control Toolkit server controls, 985–994
  Accordion control, 986–988
  Extensions, 906–911
  NoBot control, 988–990
  PasswordStrength control, 990–991
  Rating control, 991–992
  TabContainer control, 993–994
`AJAXControlExtender.vsi` **file,** 931–932
`AJAXToolkit.resources.dll`**,** 935
aliases, listing of current, 1106
`All` **attribute, Forms Authentication,** 1429
`<allow>` **node**
  authenticating/authorizing groups, 1003
  authenticating/authorizing HTTP transmission, 1003
  authenticating/authorizing users, 1000–1001
  overview of, 1001–1102
  URL Authorization, 1431–1432
`allowAnonymous` **attribute, personalization,** 740
`AllowClose` **property,** `WebPart` **class,** 848, 850
`allowDefinition` **attribute, locking-down configuration,** 1433
`AllowDirectoryBrowsing` **property, File System Editor,** 1555
`AllowLayoutChange` **attribute, Web Parts,** 834
`allowOverride` **attribute, locking-down configuration,** 1433
`AllowReadAccess` **property, File System Editor,** 1555

`AllowReturn` **attribute, Wizard server control,** 180
`AllowScriptSourceAccess` **property, File System Editor,** 1555
`AllowWriteAccess` **property, File System Editor,** 1555
AlternatingItemTemplate, DataList server control, 407–409
AlwaysVisibleControlExtender control, 937–939
animations, extender controls, 939–941, 983–984
anonymous authentication, 604–605
anonymous identity, 1430
anonymous personalization. *See* personalization, anonymous
anonymous users, URL Authorization, 1431
`AnonymousID` **property, anonymous identifier storage,** 738–739
`AnonymousIdentification_Creating` **event,** 739
`<anonymousIdentification>` **section, configuration,** 1430
<AnonymousTemplate>, LoginView control, 793–794
ANTS profiler, Red Gate Software, 1589
Apache Web Server, and `mod_rewrite`**,** 1599
`-app` **command,** 1449
`\App_Browsers` **folder,** 39
`\App_Code` **folder,** 33–38, 1297–1298
`\App_Data folder`**,** 38
`\App_GlobalResources` **folder,** 39
`\App_LocalResources` **folder,** 39
`\App_prefix`**, reserved folders,** 1571
`\App_Themes` **folder,** 38, 268–269
`\App_WebReferences` **folder,** 39
Appearance section, Web Parts, 831
Application Configuration dialog, IIS, 1027–1030
application configuration files, 1413–1414
`@Application` **directive,** `Global.asax` **file,** 52–54
application event logs, writing to, 1465–1467
Application Extension Configuration dialog, IIS, 6, 1288
application frameworks
  adding `Global.asax` file, 51–54
  application location options, 1–6
  build providers, 44–51
  compilation, 40–44
  folders, 33–39
  working with classes through VS2008, 54–61
`Application` **object,** 1059
application packaging and deploying, 1529–1566
  Custom Actions Editor, 1562–1564
  deployment pieces, 1530
  File System Editor, 1554–1557
  File Types Editor, 1559–1561
  installation programs, 1539–1547
  Launch Conditions Editor, 1568–1569
  overview of, 1529
  precompiled Web applications, 1537–1539
  Registry Editor, 1557–1559
  steps to take before, 1530–1531
  summary review, 1569–1570
  User Interface Editor, 1561–1562
  VS Copy Web Site, 1534–1537
  Windows Installer, 1547–1554
  XCOPY, 1531–1534
application pools, IIS Manager, 570–574, 1517
Application Restarts performance counter, 1469
Application Settings, IIS Manager, 1522–1523
Application tab, Web Site Administration Tool, 1510–1512
`Application_onError` **handler,** 1135
`applicationHost.config` **file, IIS Manager**
  adding Web site, 576
  delegation, 584
  hierarchical configuration, 579–581
  overview of, 572–574
`ApplicationName` **property,** `MembershipProvider` **class,** 641–644
`ApplicationProtection` **property, File System Editor,** 1555
applications
  error and exception handling, 1136–1137
  storing settings specific to, 1440–1441
  tracing, 1108–1112, 1476
Applications Running performance counter, 1469
Applied Rules list, CSS Properties Tool window, 888–891
Apply Styles tool window, Visual Studio, 888
`AppMappings` **property, File System Editor,** 1555
`appRequestQueueLimit` **attribute, runtime,** 1438
AppScaler, 1057
`<appSettings>`, `web.config` **file,** 1416–1417, 1440–1441, 1510–1511
Aptana Studio, JavaScript IDE, 1588
architecture, IIS, 7
  extensible, 562
  modular, 557–558
`.ascx` **file extension, user controls,** 1194
`.ashx` **file extension, HttpHandlers,** 1290
ASMX (ASP.NET) Web Services, 1360
`.asmx` **file extension, Web service files,** 1328, 1337–1338
ASP (Active Server Pages), 63
`<asp:MenuItemBinding>` **elements,** 702
ASP.NET
  AJAX. *See* AJAX (Asynchronous JavaScript and XML)
  application packaging and deploying. *See* application packaging and deploying

ASP.NET *(continued)*
- configuration settings, 1433–1435
- Development Server vs. IIS, 1124–1125
- Extensions Preview, 1621–1623
- forms-based authentication, 1570
- IIS 7 integrated pipeline with, 562–564
- localization threads, 1383–1386
- Machine account, 167
- Management Objects, 1441–1448
- migrating from 1x to 2.0, 1568–1570
- migrating from 2.0 to 3.5, 1570
- runtime configuration settings, 1436–1438
- Snap-In for IIS 6.0 or Windows Vista's IIS Manager, 1452–1453
- TreeView Line Generator dialog, 686
- Web Site Administration Tool. *See* Web Site Administration Tool
- Web Site Configuration Tool, 625–626
- worker process configuration settings, 1438–1440

ASP.NET MMC snap-in
- configuring providers in, 625–626
- migrating from 1x to 2.0, 1568–1570
- security through IIS, 1031

ASP.NET SQL Server Setup Wizard
- command-line tool, 591–593
- configuring personalization framework, 747
- configuring SQL-backed session state, 1053
- defined, 591
- as GUI tool, 593–594

`aspnet_compiler.exe` **tool,** 41–43

`aspnet_isapi` **extension module, IIS,** 7, 564

`aspnet_regiis.exe` **tool,** 1440, 1449

`aspnet_regsql.exe` **tool,** 591–594, 1050–1055, 1087–1092

`AspNet_SqlCacheTablesForChangeNotification` **table,** 1093

`aspnet_state.exe` **tool,** 1046–1047

`aspnet_wp.exe`**,** 1047, 1058

`ASPNETDB.mdf` **file,** 745–746, 761–775

`AspNetSql2005ProfileProvider`**,** 748–749

`AspNetSqlProfileProvider`**,** 775

ASPX Edit Helper for Visual Studio, 1595

`.aspx` **extension**
- caching with master pages, 259
- content page basics, 231–233
- content page coding, 235–239
- converting ASP.NET 1.x in Visual Studio 2008, 1578

assemblies
- adding to installer output, 1556–1557
- LINQ, 464–465
- private, 1312–1313, 1322–1323
- public, 1313–1314, 1323

`assemblies` **attribute, compilation,** 1423

`@Assembly` **directive,** 13, 22, 52–53

Assembly Registration Tool (`regasm.exe`**),** 1319, 1322–1323

assembly resource (.resx) files, 712

`AsyncCallback` **class,** 435

asynchronous command execution, 432–454
- with `AsyncCallback` class, 435
- callback approach to, 451–453, 1227–1231
- canceling, 453
- consuming Web services, 1357–1360
- `IAsyncResult` interface and, 434–436
- methods of `SqlCommand` class, 433–434
- overview of, 432–433
- poll approach to, 436–439
- using multiple wait handles, 442–451
- wait approach to, 439–442
- `WaitHandle` class, 435

Asynchronous JavaScript and XML. *See* AJAX (Asynchronous JavaScript and XML)

asynchronous postbacks
- UpdatePanel control, 918–921
- using Timer control for, 916–917

`Asynchronous` **property,** `SqlConnection` **class,** 454

AsyncPostBackTrigger, 920

atomization, 513

Attach to Process, Debug menu, 1125

Attribute Selectors, CSS, 868

attributes
- `@Assembly` directive, 22
- `@Control` directive, 18–19
- `@Implements` directive, 21
- `@Master` directive, 17–18
- `@MasterType` directive, 23
- `@OutputCache` directive, 23
- `@Page` directive, 14–16
- `@PreviousPageType` directive, 22
- `@Reference` directive, 24
- `@Register` directive, 22
- `@WebService` directive, 1329
- `<add />`, 1483
- `<browser>`, 1425
- `<compilation>`, 1422–1423
- `<eventMappings>`, 1481
- `<form>`, 760–761
- `<processModel>`, 1439–1440
- <rolemanager>, 797–798
- `<rules>`, 1484–1485
- `<sessionState>`, 1418
- `<siteMapNode>`, 663
- buffering Web events, 1492
- code-behind pages, 12
- Forms Authentication, 1429–1430
- locking-down configuration, 1433
- modifying file and directory, 1158–1160
- modifying provider behaviors, 628–633
- `<processModel>` element, 1438–1439

  - rendering HTML tag, 1214–1217
  - runtime settings configuration, 1436–1438
  - server control, 1209–1211
  - SqlMembershipProvider, 775
  - storing application-specific settings, 1440
  - XML document, 499–500
- **Audit Failure Events Raised performance counter,** 1469
- **Audit Success Events Raised performance counter,** 1469
- **Australia, culture codes,** 1382
- `Authenticate()` **method, forms authentication,** 1011
- **authentication,** 758–793
  - applying measures, 996–997
  - authorization vs., 996
  - configuring, 1427–1430
  - configuring `WindowsTokenRoleProvider`, 604–605
  - credentials, 776–784
  - defined, 758
  - forms-based. *See* Forms-based authentication
  - IIS 6 interaction with ASP.NET, 562–563
  - `MembershipProvider` class, 647–648
  - Passport. *See* Passport authentication
  - passwords and, 788–793
  - setting `<authentication>` node, 997–998
  - showing number of users online, 786–788
  - of specific files and folders, 1016–1017
  - users, adding, 761–775
  - users, authenticating, 999–1001
  - users, working with authenticated, 784–786
  - using ASP.NET MMC snap-in, 1031
  - using Security Setup Wizard, 1502–1507
  - Web site setup for, 758–761
  - Windows-based. *See* Windows-based authentication
  - `WindowsIdentity` object and, 1020–1023
- **author blogs, online resources,** 1627
- `Author` **property, Windows Installer,** 1553
- **authorization,** 793–809
  - applying to single file, 1016–1017
  - authentication vs., 996
  - configuration settings, 1430–1432
  - defined, 758
  - with LoginView server control, 793–795
  - roles, adding and retrieving application, 799–801
  - roles, adding users to, 802–803
  - roles, caching of, 807–809
  - roles, checking for specific user in, 806–807
  - roles, deleting, 801–802
  - roles, getting all users of particular, 803–804
  - roles, getting for specific user, 805
  - roles, removing users from, 805–806
  - roles, setting up Web site for managing, 796–799
  - of specific files and folders, 1017
  - Windows-based, 999–1003
- **authorization, programmatic,** 1017–1023
  - overview of, 1017–1018
  - with `User.Identity` property, 1018–1019
  - using `User.IsInRole()` method, 1019–1020
  - with `WindowsIdentity`, 1020–1023
- `<authorization>` **element,** 776
- `AuthorizationStoreRoleProvider`, 605–606
- **Auto Format, TreeView server control,** 675
- **AutoCompleteExtender control,** 941–944
- `AutoDetect` **value, storing identifiers,** 738
- `autoEventWireup` **element, page configuration,** 1434
- `AutoGenerateDeleteButton` **property,** 341, 349–350
- `AutoGenerateEditButton` **attribute,** 336
- `automaticSaveEnabled` **attribute,** `<profile>` **node,** 744–745
- `AutoPostBack` **property**
  - DropDownList control, 123
  - RadioButton server control, 136
  - TextBox server control, 113–114

B

- **Base Class Library (BCL),** 1139
- **Basic authentication,** 1004–1005
- `batch` **attribute, compilation configuration,** 1422
- **Batch section, .NET compilation,** 1517–1518
- `batchTimeout` **attribute, compilation configuration,** 1422
- **BCL (Base Class Library),** 1139
- **before element, CSS,** 871
- `BeginExecuteReader`, **asynchronous commands,** 436–439
- ***Beginning CSS: Cascading Style Sheets for Web Design, 2nd Edition* (York),** 862
- ***Beginning JavaScript, Third Edition* (Wiley Publishing, Inc.),** 89
- **Beyond Compare,** 1601
- `Bin` **folder,** 1301
- **binary metabase,** 1410
- `BinaryReader` **class,** 1173–1174
- `BinaryWriter` **class,** 1173–1174
- `Bind` **method, data binding,** 368
- **binding**
  - early vs. late, 1320
  - Menu server control to XML file, 701–702
  - TreeView control to XML file, 676–679
- **Blend, Microsoft,** 1618–1619
- **block boxes,** 877–879

blogs (weblogs), and `XmlDataSource` control, 535–537
blogs (weblogs), online resources, 1627
Blowery Http Compression Module, 1178
`<body>` **tag,** 273
`BooleanSwitch`, 1119–1120
bootstrapping, 1552
borders, CheckBoxList control, 133
Bound List controls
  AdRotator control, 366
  DetailsView server control, 344–350
  DropDownList control, 365
  FormView, 360–365
  GridView control. *See* GridView control
  ListBox control, 365
  ListView control. *See* ListView server control
  Menu control, 366
  overview of, 317–318
  RadioButtonList control, 365
  TreeView control, 366
box model, CSS, 877–880
breadcrumb navigation, SiteMapPath control, 664
.`browser` **files,** 1231–1233, 1424
browsers
  Administrative Tool and, 1470–1476
  `\App_Browsers` folder, 39
  ASP.NET server controls and, 64
  client-side caching and, 1078–1080
  configuration settings, 1423–1425
  container-specific master pages and, 257–258
  displaying event log contents, 1462–1464
  downlevel, 1231–1234
  implementing CSS, 862
  internal stylesheets and, 75
  limitations of classic ASP, 63–64
  state management and, 1034
  supporting Basic authentication, 1004
`buffer` **element,** 1434
buffer uploads, 1437
`BufferedWebEventProvider`, 613, 617, 619, 1490–1492
buffering, Web events, 1490–1492
Build Calculator, 1299
build providers, 44–51
  compilation configuration, 1423
  constructing own, 46–51
  overview of, 44–45
  using built-in, 45–46
BulletedList server control, 157–162
business objects, .NET
  COM Interoperability with. *See* COM Interop
  defined, 1297
  using from unmanaged code. *See*.NET from unmanaged code
  using in ASP.NET 3.5, 1298–1302
Button server control
  `CausesValidation` property, 115
  `CommandName` property, 115–116
  ImageButton server control, 119–120
  LinkButton server control, 119
  overview of, 115–118
  working with client-side JavaScript, 117–118
`Button_Command` **event,** 116–117
buttons
  events, 68–70
  Wizard navigation, 183
`Byte` **arrays,** 173–174, 1168

C

Cab Project, 1541
CAB Size, 1553
cabinet files, 1553
`Cache` **object**
  `Application` object vs., 1059
  attaching SQL Server cache dependencies to, 1097–1100
  cache dependencies, 1081–1087
  data caching using, 1080–1081
Cache-Control HTTP Header, 1078–1080
`CacheDependency`
  `AggregateCacheDependency` class, 1082
  defined, 1082
  setting dependencies with `Insert` method, 1081
  `SqlCacheDependency` class, 1087, 1093, 1096–1098
  `UnsealedCacheDependency` class, 1082–1083
  writing own, 1083–1087
`CacheDuration` **property, SOAP responses,** 1350–1351
caching, 1071–1101
  ASP.NET applications, 1092–1093
  data source control, 314–316
  defined, 1071
  `HttpCachePolicy` class and client-side, 1078–1080
  with master pages, 259
  output, 1071–1074
  overview of, 1071
  partial page, 1074–1075
  post-cache substitution, 1075–1077
  programmatically, 1080–1087
  roles, 807–809
  testing SQL Server invalidation, 1094–1101
  of user control to its native type, 1200–1201
  using AnimationExtender control, 944
  using SQL Server Cache Dependency, 1087–1092
  Web service responses, 1349–1350
Calendar server control, 142–151
  date format/ranges, 144–147
  date selection, 142–143
  day, week or month selections, 144
  overview of, 142
  style and behavior, 147–151
CalendarExtender control, 944–946
Canada, culture codes, 1382
`Cancel` **method, asynchronous processing,** 453
CAPI (cryptographic API), 623

`Caption` **attribute, Table server control,** 140–141
Cascading Style Sheets. *See* CSS (Cascading Style Sheets)
Cassini built-in Web server, 2, 1124
Catalog Mode, Portal Framework Web pages, 812, 825–826, 828
`CausesValidation` **property, Button server control,** 115
CCW (Com-Callable Wrapper), 1314–1316
`.cd` **file extension (class designer),** 54–60
`Cell` **property,** `DayRenderEventArgs` **class,** 150
ChangeConflictException, 493
ChangePassword server control, 788–789
check boxes, in TreeView, 679–683
CheckBox server control
  CheckBoxList server control vs., 132
  overview of, 129–132
  RadioButton server control vs., 134
  with ToggleButtonExtender control, 982–983
  using MutuallyExclusiveCheckBoxExtender control with, 967–968
CheckBoxList server control, 124–125, 132–134
`Checked` **property, CheckBox server control,** 131
child elements, SiteMapPath server control, 670
child nodes, TreeView control, 676
Child Selectors, CSS, 868
Choose Location dialog, 2–6
Choose Toolbox Items dialog, Visual Studio 2008 Toolbox, 932
class designer (`.cd` **file extension),** 54–60
Class Selectors, CSS, 869
classes
  \App_Code folder for, 33–38
  basic ADO.NET, 383–384
  configuration file programming, 1441–1442
  CSS pseudo, 869–871
  derived from `Stream` class, 1167
  naming in LINQ to SQL, 482
  Portal Framework, 841–844
  working with, 54–61
classic ASP
  ASP.NET today vs., 63–64
  event model, 67–68
  `include` files in, 229–231
  `Session` object in, 1036
Client Script Library, AJAX, 900–901
`ClientID` **property, server controls,** 1217
`ClientScriptManager` **class,** 1222–1227
client-side, adding features to server controls, 1222–1231
  accessing embedded resources, 1227
  asynchronous callbacks, 1227–1231
  emitting client-side script, 1222–1227
client-side AJAX, 900–902
client-side authentication settings, 1427
client-side caching, 1078–1080
client-side callback, 89–104
  complex example of, 99–104
  overview of, 89
  simple approach for, 90–96
  typical postback vs., 89
  using single parameter, 96–99
client-side culture declarations, 1387–1389
client-side Javascript debugging, 1131–1134
client-side state management, 1035
client-side validation
  CustomValidator control with, 211–213, 216
  server-side validation vs., 194–195
  turning off, 220–221
`Close`**( ) method**
  `Stream` class, 1168–1170
  `XmlReader`, 525, 548–549
  `XmlWriter`, 525
CLR objects, 515–518
`cmd.ExecuteNonQuery( )` **command, ADO.NET,** 381
code
  content pages, 235–239
  in-line pages, 6–10
  master pages, 233–235
  online converter tool for, 1600
  server controls, 66–67
  tools for tidying up, 1591–1594
Code Converter, online, 1600
Code Style Enforcer, 1592
code-access security (CAS), 1412–1413
code-behind pages
  for AutoCompleteExtender, 942–943
  creating new page using, 10–12
  for CustomerOrders page, 430–431
  for DynamicPopulateExtender control, 957–959
  for NoBot control, 989–990
  overview of, 6–8
  for XML Web, 1329–1330
`CodeExpression` **class,** 371
`CodeFile` **attribute,** `Page` **directive,** 12
`codegen` **folder, buffer uploads,** 1437
`codeSubDirectories` **attribute, compilation configuration,** 1423
`CollapseAll( )` **method,** `TreeView` **class,** 687–690
`Collapsed` **property, CollapsiblePanelExtender control,** 947
CollapsiblePanelExtender server control, 946–947
collections, visually removing items from, 124–125
Color Picker, 1271
columns
  DataList control, creating multiple, 409–410
  Grid View control, sorting, 323–325
  GridView control, customizing, 328–331
  GridView control, editing row data, 334
  GridView control, using TemplateField, 331–334
  inserting XML data into, 554–556
COM Interop, 1302–1313
  assessing tricky members in C#, 1308–1309

COM Interop ***(continued)***
  deploying components with .NET apps, 1312–1313
  error handling, 1309–1312
  overview of, 1302–1303
  releasing objects manually, 1309
  Runtime Callable Wrapper, 1303–1304
  using objects in ASP.NET code, 1304–1308
Com-Callable Wrapper (CCW), 1314–1316
Command and Parameter Editor dialog, 295
`Command` **object**
  defined, 383–384
  using `DataReader` object, 387–389
  using in ADO.NET, 386–387
Command window, 1106
`CommandField`**,** 336
`CommandName` **property, Button server control,** 115–116
comments
  internal stylesheets using HTML, 75
  in XML documents, 499
Comments view, Task List, 1106–1107
CompareValidator server control, 196, 202–206
compilation
  `<pages>` configuration, 1435
  configuration, 1421–1423
  configuration settings, 1421–1423
  IIS Manager, 1517–1518
  overview of, 40–44
  precompilation for deployment, 1537–1539
  precompilation of .NET business objects, 1298–1301
components, tracing from, 1113–1114
composite controls, 1244–1247, 1260–1262
compression, deployment project, 1552
compression, streams
  HTTP output, 1178–1181
  using `Deflate()` method, 1177–1178
  using `GZipStream`, 1176–1177
Computer Management utility, 1002–1003
concurrency, LINQ to SQL, 493
`<configSections>` **element,** 903–904, 1416
`configSource` **attribute, include files configuration,** 1435
configuration, 1409–1460
  altering SQL Server used via, 596–597
  anonymous identity, 1430
  application file, 1413–1414
  applying in hierarchal manner, 996
  ASP.NET pages, 1433–1435
  ASP.NET runtime settings, 1436–1438
  ASP.NET worker process, 1438–1440
  authentication, 1427–1430
  authorization, 1430–1432
  browser capabilities, 1423–1425
  compilation, 1421–1423
  connecting strings, 1416–1417
  custom errors, 1426–1427
  custom sections, 1453–1460
  detecting changes, 1415
  editing files, 1452–1453
  file format, 1415–1416
  health monitoring, 1479–1486
  with IIS Manager. *See* IIS (Internet Information Service) Manager
  include files, 1435
  locking-down settings, 1433
  overview of, 1409–1410
  programming files, 1441–1448
  protecting settings, 1448–1452
  of providers, 625–626
  server configuration files, 1411–1413
  session state, 1417–1421
  settings application, 1414–1415
  storing application-specific settings, 1440–1441
  summary review, 1460
  with Web Site Administration Tool. *See* Web Site Administration Tool
  writing events via, 1486–1487
configuration providers, 620–623
Configure Behavior dialog, 492
Configure Data Source Wizard, 289–293, 299–301
Configure Data Wizard, 307–309
Configure ListView dialog, 352–353
ConfirmButtonExtender control, 947–948
`ConflictDetection` **property, SqlDataSource control,** 296–297
`Connection` **object**
  defined, 383
  executing SQL queries with, 386–387
  using `DataReader` object, 387–389
  using in ADO.NET, 384–386
connection strings
  configuration settings, 1416–1417
  connecting `SqlProfileProvider` to SQL Server 2005, 607–608
  creating `ActiveDirectoryMembershipProvider`, 600
  enumerating, 1443–1445
  IIS Manager, 1523–1524
  protecting configuration settings, 1448–1452
  session state configuration, 1420–1421
  storing connection information, 315–317
  using `Connection` object in ADO.NET, 384–386
  using `ConnectionStringBuilder` for, 317
connections
  asynchronous, 454
  to data source, 419–422
  to Oracle Database, 400–403
  SqlDataSource control, 289–293
  between WebParts. *See* Web Parts, connecting
Connections pane, IIS Manager, 569–576

- `ConnectionString` **property, SqlDataSource control,** 292–294, 296, 300, 746
- `ConnectionStringBuilder` **class,** 317
- `connectionStringName` **attribute,** `<add>` **element,** 1092–1093
- **constants, validating against,** 204–206
- **consumer Web Parts,** 850–851, 854–858
- **consuming XML Web services.** ***See*** **XML Web services, consuming**
- `Container.DataItem` **syntax,** 368
- **containers, ListView,** 356–357
- **container-specific master pages,** 257–258
- **content areas,** 231–233, 250–251
- **content pages**
  - basics, 231–233
  - caching with master pages, 259
  - coding, 235–239
  - mixing languages/master pages with, 239–241
  - programmatically assigning master pages to, 251–253
  - working with page title on, 242–243
- `ContentPlaceholder` **server control,** 250–251, 253–256
- `<ContentTemplate>`**, LoginView control,** 795
- `<ContentTemplate>`**, UpdatePanel server control,** 917–918
- `ContinueButtonClick( )` **event, CreateUserWizard control,** 765–766
- `Control` **class,** 70
- **control designers,** 1258–1271
  - designer actions, 1269–1271
  - designer regions, 1259–1269
  - overview of, 1258
- `@Control` **directive**
  - defined, 13
  - overview of, 18–19
  - for user controls, 1194
- **control IDs,** 1217
- `ControlContainer` **class,** 1258
- **controls**
  - data source. *See* data source controls
  - IIS Manager, 1524
  - master page, 243–249
  - server. *See* server controls
  - styling ASP.NET, 891–893
- `Controls` **collection,** 1198–1200
- **ControlState**
  - client-side state management, 1036
  - state management and, 1067
  - using in server controls, 1236–1238
- `ControlToValidate` **property**
  - CompareValidator control, 203
  - CustomValidator control, 213
  - Regular Expression validator control, 210
  - RequiredFieldValidator control, 198–199, 201–202
- **conversion, currency,** 1392–1394
- `ConversionReport.webinfo` **file,** 1578–1579
- `cookieless` **attribute, Forms Authentication,** 1429
- **cookieless session state,** 1057–1058, 1418, 1526
- `cookieName` **attribute,** 737, 1041
- **cookies**
  - achieving identification without, 738
  - changing length of time stored, 737–738
  - changing name for anonymous identification, 737
  - client-side state management, 1034–1036
  - with Login control, 778
  - persistent user data storage and, 723
  - roles cached in, 807–809
  - setting anonymous, 737
  - storing only non-sensitive information in, 1060
- `cookieTimeout` **attribute, anonymous personalization,** 737–738
- **Copy Web Site GUI, deployment option,** 1534–1537
- **copy-and-paste deployment, XCOPY,** 1531–1534
- `Corners` **property, RoundedCornersExtender control,** 976
- **counters, performance,** 1468–1476
- **CR_Documentor,** 1603
- `Create( )` **method,** `XmlReader`**,** 506–508
- **Create or Manage Roles link, ASP.NET Web Site Administration Tool,** 715–716
- `CreateChildControls( )` **method, Web Parts,** 849, 854, 856
- `CreatedUser( )` **event, CreateUserWizard control,** 765–766, 768–769
- `CreateEventSource()`**, writing to event log,** 1466
- `CreateNewStoreData( )` **method,** `SessionStateModule`**,** 1056
- `CreateRole( )` **method,** `Roles` **class,** 653–655, 658–659, 800
- `createSilverlight( )` **method,** 1615–1616
- `CreateUser( )` **method, Membership API,** 770–772
- **CreateUserWizard server control**
  - adding user to membership service, 761–763
  - using personalization properties, 766–770
  - working with, 765–766
- **credentials**
  - authenticating against values in database, 1012–1014
  - authenticating against values in `web.config` file, 1010–1012
  - Basic authentication, 1004–1005
  - Digest authentication, 1005–1006
  - locking out users with bad passwords, 780–784
  - logging in users programmatically, 779–780
  - remote debugging, 1126–1128

credentials *(continued)*
  turning off access with <authorization> element, 776
  using Login server control, 776–779, 1014–1015
cross-page postbacks
  Button server control, 118
  overview of, 27–33
  state management and, 1061–1063
cryptographic API (CAPI), 623
cryptographic service provider (CSP), 623
`CsaProviderName` **attribute,** `RsaProtectedConfigurationProvider`**,** 623
CSP (cryptographic service provider), 623
CSS (Cascading Style Sheets)
  changing styles using, 72–75
  creating files for themes, 272–275
  designing. *See* HTML and CSS design
  incorporating images into themes using, 276–278
  references in ResizableControlExtender control, 973–974
  styling HTML for server controls, 1217–1220
CSS properties grid, CSS Properties Tool window, 888–891
CSS Properties tool window, Visual Studio, 888–891
`CssClass` **property, ASP.NET server controls,** 892–893
`culture` **attribute,** `Web.config` **file,** 711
`culture` **attribute,** `web.config.comments` **file,** 1386–1387
culture declarations
  client-side, 1387–1389
  server-side, 1386–1387
culture types
  localization, 1382–1383
  .NET globalization, 1518
`CultureInfo` **object, ASP.NET threads,** 1383–1386
currency, localization, 1391–1394
`CurrentNode` **property,** `SiteMap` **class,** 706–709
Custom Actions Editor, 1562–1564
custom sections, configuration, 1453–1460
custom server controls, 281–285
`<customErrors>` **section, configuration settings,** 1426–1427
Customize Line Images, TreeView control, 685–686
CustomProviders, 635–636
CustomValidator server control, 196, 211–216
`CutoffMaximumInstances` **property, NoBot control,** 989
`CutoffWindowSeconds` **property, NoBot control,** 989

D

data binding, 287–375
  expressions and expression builders, 369–374
  inline data-binding syntax, 367–369
  ListView, 357–359
  using data source controls. *See* data source controls
data caching, 1080–1081
data concurrency, 297, 306–307
data context, LINQ to SQL, 483–484
data contract, WCF service, 1364, 1374–1377
data item rendering, ListView, 355–356
data management, with ADO.NET. *See* ADO.NET
Data Protection API (or DPAPI), 1448–1452
`Data Provider` **classes, ADO.NET,** 383
data providers. *See* providers
data source controls
  AccessDataSource control, 307
  configuring caching, 314–316
  LinqDataSource control, 302–307
  ObjectDataSource control, 309–313
  overview of, 288
  SiteMapDataSource control, 314
  `SqlDataSource` control. *See* `SqlDataSource` control
  storing connection information, 315–317
  using Bound List controls with. *See* Bound List controls
  XmlDataSource control, 307–309
data source, creating connection to, 419–422
data stores, 38, 161–162
data visualizers, debugging using, 1129
`DataAdapter` **object,** 383–384, 389–392
databases, 544–556
  authenticating against values in, 1012–1014
  debugging, 1134
  enabling for SQL Server cache invalidation, 1088
  routing events to SQL server, 1487–1490
  SQL Server cache invalidation, 1091
  XML and, 544–549
  XML data type and SQL Server 2005, 549–556
`DataBind( )` **method, list controls,** 318
data-binding, GridView control, 333–334
`DataFile` **property, AccessDataSource control,** 307
DataGrid control, 323, 325–326
`DataKeyNames` **property, GridView control,** 336–337
DataList server control, 403–410
  available templates, 403–404
  ItemTemplate, 404–407
  multiple columns, 409–410
  other layout templates, 407–409
  styling with CSS-Friendly Control Adapters, 893
DataPager control, 359–360
`DataReader` **object**
  defined, 383–384

  - loading `DataTable` from, 396–398
  - reading data from SQL database, 378–379
  - using `DataSet` vs., 398–399
  - using in ADO.NET, 387–389
- **Dataset Designer view,** 505
- **DataSets**
  - building, 427–432
  - defined, 395
  - overview of, 530
  - persisting to XML, 530–531
  - typed, 400
  - when to use, 398–400
  - `XmlDataDocument`, 531–533
- `DataSourceMode` **property, SqlDataSource control,** 293
- `DataTable` **objects,** 395–400, 530–531
- **datatips, debugger,** 1128–1129
- **dates**
  - Calendar control. *See* Calendar server control
  - CalendarExtender control, 944–946
  - localization, 1389–1391
- `Day` **property,** `DayRenderEventArgs` **class,** 150–151
- `DayRender` **event, Calendar control,** 147–150
- **DbgView, debugging using,** 1113
- **DCOM,** 1341
- `debug` **attribute, compilation configuration,** 1422
- **debugging,** 1122–1134
  - in Application tab, 1511–1512
  - client-side Javascript, 1131–1134
  - datatips, 1128–1129
  - debug vs. release, 1123
  - design-time support for, 1103–1107
  - IIS vs. ASP.NET Development Server, 1124–1125
  - JIT dialog and, 1123–1124
  - new tools for, 1128–1131
  - overview of, 1122–1123
  - remote, 1126–1128
  - SQL stored proc and, 1134
  - starting session of, 1125–1128
  - tools, 1583–1589
  - tracing and. *See* tracing
  - turning off before deploying, 1530–1531
  - XSLT, 543–544
- `Debug.Write` **function,** 1113–1114
- **declarations, culture,** 1386–1389
- **declarations, XML documents,** 499
- **DeclarativeCatalogPart control,** 834–836
- **decryption**
  - configuration settings, 1451–1452
  - ViewState, 1066–1067
- `Default.asp`**,** 1040, 1044–1046
- `Default.aspx` **page**
  - adding language to, 1400–1401
  - finalizing localization, 1401–1403
  - local resource files, 1399
  - setting up administrators' section, 717
  - user login, 651–652
- `DefaultDocument` **property, File System Editor,** 1555
- `defaultProvider` **attribute**
  - compilation configuration, 1423
  - connecting to new SQL Server instance, 595–597
  - defining provider instance, 641–642
  - installing personalization, 748
  - `SqlMembershipProvider`, 598
  - `SqlRoleProvider`, 604, 605
- `defaultRedirect`**, custom errors configuration settings,** 1427
- `defaultUrl` **attribute, Forms authentication,** 1429–1430
- `defaultValue` **attribute, personalization,** 735
- `Deflate( )` **method, streams,** 1177–1178
- **delayed execution, LINQ,** 469
- **delegation, IIS Manager,** 581–584
- `DeleteAllOnSubmit` **method, LINQ to SQL,** 493–494
- `DeleteCommand`**, DetailsView control,** 350–351
- `DeleteCookie( )` **method, Roles API,** 808–809
- `DeleteOnSubmit` **method, LINQ to SQL,** 493–494
- `DeleteProfile( )` **method,** `ProfileManager`**,** 755
- `DeleteRole( )` **method,** `Roles` **class,** 655, 801–802
- `DeleteUser( )` **method,** `ProfileManager`**,** 755
- **deleting, users from roles,** 805–806
- **deleting data**
  - from SQL Server, 382
  - using DetailsView server control, 349–350
  - using GridView control, 341–343
  - using LINQ to SQL, 493–494
  - using SqlDataSource control, 342
- **DelimitedListTraceListener,** 1118–1119
- **denial of service (DoS) attacks,** 168, 1437
- `<deny>` **node**
  - authenticating/authorizing groups, 1003
  - authenticating/authorizing HTTP transmission method, 1003
  - authenticating/authorizing users, 1000–1001
  - denying unauthenticated users, 776
  - overview of, 1001–1102
  - URL authorization, 1431–1432
- **dependencies**
  - component, 1551–1552
  - IIS 7 update, 565
- **dependencies, cache,** 1081–1087
  - `AggregateCacheDependency` class, 1082
  - creating custom, 1083–1087
  - testing SQL Server cache invalidation, 1096–1100
  - `UnsealedCacheDependency` class, 1082–1083
  - using SQL Server Cache Dependency, 1087–1092
- **deployment. *See* application packaging and deploying**
- **Descendant Selectors, CSS,** 868
- `Description` **property,** `ProviderBase` **class,** 633

- `Description` **property, Windows Installer,** 1553
- `DescriptionUrl` **attribute, Image server control,** 139
- **Design Mode, Portal Framework Web pages,** 812, 821, 826–828
- **Design view**
  - applying styles to Menu control in, 694–695
  - dragging and dropping controls in, 65–66
  - HTML and CSS in, 884–885
  - HTML server controls in, 76
  - server control events in, 68–70
  - viewing nested master pages in, 255–256
- `Designer` **attribute,** 1259, 1262
- **designer regions**
  - creating composite control with, 1260–1262
  - customizing designer class to define, 1262–1269
  - overview of, 1259
- **DesignerActionList class,** 1269–1271
- **design-time**
  - syntax notifications, 1104–1106
  - Task List views, 1106–1107
  - tracing. *See* tracing
  - using Immediate and Command window at, 1106
- **design-time, server control behaviors at,** 1254–1273
  - custom type converters, 1256–1258
  - designers, 1258–1271
  - overview of, 1254
  - type converters, 1254–1255
  - UI type editors, 1271–1273
- **desktop shortcuts,** 1557
- **DetailsView server control,** 344–350
  - CSS-Friendly Control Adapters for styling, 893
  - customizing display of, 345
  - with GridView control, 345–349
  - inserting, updating and deleting data, 349–350
  - overview of, 344–345
  - `SelectParameters` vs. `FilterParameters`, 349
- `DetectNewerInstalledVersion` **property, Windows Installer,** 1553
- **developers**
  - ASP.NET AJAX and, 901–902
  - tools for, 1600–1604
- **diagnostic switches,** 1119–1121
- **dialog boxes, reporting validation errors,** 220
- **dialogs, adding to installation process,** 1562
- `DictionarySectionHandler`**,** 1453, 1456–1457
- **DiffMerge,** 1601
- **Digest authentication,** 1005–1006
- `Direction` **attribute, Panel server control,** 155–156
- **directives**
  - `Global.asax` file, 52–53
  - page. *See* page directives
- **directories,** 1147–1148
  - browsing with `Directory`/`DirectoryInfo`, 1143–1148
  - displaying files using `File`/`FileInfo`, 1149–1154
  - I/O shortcuts, 1175–1176
  - modifying Access Control Lists on, 1160–1166
  - modifying properties and attributes, 1158–1160
  - setting/getting current working, 1147–1148
  - working with `DriveInfo` class, 1140–1143
- `Directory` **class,** 1143–1148
- `DirectoryInfo` **class**
  - browsing directory structure with, 1143–1148
  - displaying files in directory, 1149–1154
  - modifying file and directory properties with, 1159–1160
- `Disconnected` **classes, ADO.NET,** 383
- `Display` **property, CSS box model,** 879–880
- `Display` **property, ValidatorCalloutExtender control,** 985
- `DisplayAfter` **attribute, UpdateProgress control,** 924
- `DisplayMode` **property, BulletedList control,** 159–161
- `DisplayMode` **property, ValidationSummary control,** 218–219
- `DisplayRememberMe` **property, Login control,** 776–777
- `Dispose` **method, IHttpModule interface,** 1277–1278
- `Dispose` **statement,** 1168
- **<div> element, DragPanelExtender control,** 950–951
- **DLLs**
  - in ASP.NET applications, 1301–1302
  - localization, using AJAX controls, 936
  - precompiling business objects into, 1298–1301
- **DOCTYPES,** 862
- **dollar symbol ($), currency translation,** 1393–1394
- `domain` **attribute, Forms authentication,** 1430
- **domain names,** 1025–1027
- **DoS (denial of service) attacks,** 168, 1437
- **dotTrace profiler, Jetbrain,** 1589
- **DPAPI (Data Protection API),** 1448–1452
- `DpapiProtectedConfigurationProvider`**,** 620
- **drag and drop, server controls,** 65–66
- **DragPanelExtender control,** 950–951
- `DriveInfo` **class,** 1140–1143
- **drives,** 1140–1143
- `DriveType` **enumeration,** 1141–1142
- **drop-down lists, validating,** 201–202
- **DropDownExtender control,** 951–953
- **DropDownList control**
  - adding list of modes to Web page, 822–825
  - building `ProfileManager.aspx` page with, 751–754
  - as databound control, 365
  - reading from event log, 1462–1464
- **DropDownList server control**
  - ListBox server control vs., 125–126
  - overview of, 121–124

  using ListSearchExtender control, 962–964
  visually removing items from collections, 124–125
DropShadowExtender control, 953–955
`Duration` **attribute, output caching,** 1072
dynamic compilation configuration, 1421–1423
dynamic links, 695, 697
dynamic pseudo classes, CSS, 870–871
`DynamicBottomSeparatorImageUrl` **property, Menu control,** 699–700
`DynamicPopOutImageUrl` **property, Menu control,** 699
DynamicPopulateExtender control, 956–959
`DynamicTopSeparatorImageUrl` **property, Menu control,** 699–700

E

early-bound access, personalization, 729–730
Edit and Continue, debugging, 1130
Edit and Refresn, debugging, 1130
Edit Columns, GridView control, 329
Edit Mode, Portal Framework Web pages, 812–813, 828–833
Edit Templates, GridView, 322–323
editing
  configuration files, 1452–1453
  in GridView. *See* GridView control, editing
EditItemTemplate, ListView, 359, 415
editor, resource, 1406–1407
`Editor` **attribute, UI type editors,** 1271–1273
Editor Zone, adding to page, 829–830
editors
  Custom Actions Editor, 1562–1564
  File System Editor, 1554–1557
  File Types Editor, 1559–1561
  Launch Conditions Editor, 1568–1569
  Registry Editor, 1557–1559
  User Interface Editor, 1561–1562
EditTemplate, GridView control, 334
ELMAH (Error Logging Modules and Handlers), 1598
e-mail
  notification, 615–618
  password recovery using, 789–792
  sending from Web page, 1189–1190
  SMTP E-mail, 1526–1527
  Web events, 1492–1497
embedded resources, accessing, 1227
empty elements, XML, 501
`EmptyDataTemplate` **property, GridView,** 321–323
`EmptyDataText` **property, GridView,** 321–323
`EmptyItem` **template, ListView,** 357, 415–416
Enable 32-bit Applications, IIS Manager, 573
`enable` **attribute, runtime settings configuration,** 1436
`EnableCaching` **property, AnimationExtender control,** 944
`EnableDelete` **property, LinqDataSource control,** 304
`EnableInsert` **property, LinqDataSource control,** 304
`enableLocalization` **attribute,** `<siteMap>` **element,** 710
`EnableSessionState` **attribute,** `<pages>` **configuration,** 1434
`EnableSessionState` **attribute,** `Session` **performance,** 1045–1046
`EnableSortingAndPagingCallbacks` **property, GridView,** 327
`EnableTheming` **attribute,** 266–268
`EnableUpdate` **property, LinqDataSource control,** 304
`enableViewState` **element,** `<pages>` **configuration,** 1434
encryption
  configuration settings, 1448–1452
  Digest authentication with, 1005–1006
  Forms-based authentication with, 1012, 1429
  ViewState, 1066–1067
`End` **events,** `Session` **object,** 1037
`EndExecuteReader`**, asynchronous commands,** 436–439
endpoints, WCF service, 1363
English-speaking culture codes, 1382
`EnsureChildControls()` **method, composite controls,** 1247
Enterprise Services, 1360
Error Events Raised performance counter, 1469
error handling
  <customErrors> syntax for, 1426–1427
  adding users programmatically, 772–773
  configuration settings, 1427
  design-time support for, 1103–1107
  health monitoring and. *See* health monitoring
  in .NET, 1309–1312
  in .NET from unmanaged code, 1320–1322
  overview of, 1134–1138
  SQL-backed session state, 1054
  tracing and. *See* tracing
  using data source control events for database, 298–299
Error Logging Modules and Handlers (ELMAH), 1598
error notifications
  debugging using, 1130
  design-time support for, 1103–1104
  e-mail Web events, 1495–1497
  using images and sounds for, 221–222
`Error` **property,** `Page` **class,** 1135

- `ErrorMessage` **property, ValidationSummary control,** 217–218
- `Eval` **method, data binding syntax,** 368
- `EvaluateExpression` **property, expression builders,** 373–374
- **event logs**
  - health monitoring. *See* health monitoring
  - instrumentation overview, 1461–1462
  - reading, 1462–1464
  - writing, 1464–1468
- **EventLogTraceListener,** 1116–1118
- `EventLogWebEventProvider`
  - defined, 613
  - health monitoring configuration, 1479
  - overview of, 613–615
- `<eventMappings>`**, health monitoring configuration,** 1479–1482
- `eventName` **attribute,** `<rules>` **configuration,** 1484
- **events**
  - AnimationExtender control, 939–941
  - anonymous identification, 739
  - buffering Web, 1490–1492
  - CreateUserWizard control, 765–766
  - e-mailing Web, 1492–1497
  - global application, 52–54
  - GridView control, 319, 327
  - `Init` method registering, 1278
  - LinqDataSource control, 307
  - master page, 258–259
  - Menu server control, 700–701
  - ObjectDataSource control, 313
  - page, 24–26
  - postback, 1238–1242
  - `ProfileModule` class, 741
  - server control, 67–70, 1210–1211
  - sessions and, 1036–1037
  - Silverlight, 1623–1626
  - SQL server, 1487–1490
  - SqlDataSource control, 297–299
  - UpdatePanelAnimationExtender control, 983–984
  - user control, 1195–1196, 1201–1203
  - validation, 196–197
  - Web, 1121–1122
  - Wizard server control, 183–184
  - writing via configuration, 1486–1487
  - XmlDataSource control, 307–309
- **exception handling,** 1134–1138
  - application-level, 1136–1137
  - HTTP Status Codes, 1137–1138
  - methods for, 1134–1135
  - page-level, 1135–1136
- `ExceptionHandled` **property, database errors,** 298–299
- **Execute methods,** `Command` **object,** 386–387
- `ExecuteNonQuery()` **command, SQL Server,** 382
- `ExecutePermissions` **property, File System Editor,** 1555
- `executionTimeout` **property,** 169–170, 1436–1437
- `ExpandAll()` **method,** `TreeView` **class,** 687–690
- `ExpandControlID` **property, CollapsiblePanelExtender control,** 947
- `explicit` **attribute, compilation configuration,** 1423
- **Expression Blend, Microsoft,** 1618–1619
- **expression builders,** 369–374
- **expressions,** 369–374
- **extending ASP.NET, tools for,** 1597–1599
- **extensible architecture, IIS,** 7, 562
- **eXtensible Markup Language.** ***See*** **XML (eXtensible Markup Language)**
- **Extensions Preview, ASP.NET,** 3.5, 1621–1622
- **external style sheets, CSS**
  - combining with internal style sheets, 876
  - creating, 73–74
  - overview of, 863–865

F

- **F5, starting debug session,** 1125
- **Fiddler tool,** 671, 1078
- **field names, LINQ queries,** 467
- **file buffering, runtime configuration,** 1437
- **file extensions**
  - `.ascx` (user controls), 1194
  - `.ashx` (HttpHandlers), 1290
  - `.asmx` (Web service files), 1328, 1337–1338
  - `.aspx` pages, 231–233, 235–239, 259, 1578
  - `.cd` (class designer), 54–60
  - `.js` (JavaScript), 1573–1574
  - mapping in IIS, 1293–1295
  - mapping with File Types Editor, 1559–1561
  - `.master` (master pages), 231–235
  - `.mdf`. *See* SQL Server Express Edition (`.mdf` file)
  - `.msi` (Windows Installer), 1540
  - `.resx` (resource files), 1397
  - using AJAX, 1315–1316
  - `.vsi (AJAXControl Extender)`, 931–932
  - working with, 1027–1030
  - `.wsdl` (Web Services Description Language), 33–38
- **file I/O and streams,** 1140–1166
  - `Directory` and `DirectoryInfo` classes, 1143–1148
  - `DriveInfo` class, 1140–1143
  - `File` and `FileInfo` objects, 1149–1154
  - modifying Access Control Lists, 1160–1166
  - modifying properties and attributes, 1158–1160
  - network communications. *See* network communications
  - security and, 1139
  - using relative paths and setting/getting current directory, 1148–1149
  - working with paths, 1154–1158
  - working with serial ports, 1181–1182
- **file I/O and streams, reading and writing files,** 1166–1181
  - compressing streams, 1176–1181
  - encodings, 1174–1175
  - I/O shortcuts, 1175–1176

  - overview of, 1166
  - `Reader` and `Writer` classes, 1171–1174
  - streams, 1167–1171
- **File System Editor,** 1543, 1554–1557
- **File System setting, Copy Web Site,** 1535
- **File Transfer Protocol.** ***See*** **FTP (File Transfer Protocol)**
- **File Types Editor,** 1559–1561
- `FileInfo` **class,** 1149–1154, 1159–1160
- **FileIO access, server configuration,** 1413
- **files**
  - adding to installer output, 1556–1557
  - authenticating specific, 1016–1017
  - authorization of, 1432
  - configuration of. *See* configuration
  - `File/FileInfo` displaying directory, 1149–1154
  - I/O shortcuts, 1175–1176
  - modifying ACLs in, 1160–1166
  - modifying properties and attributes in, 1158–1160
- `FileStream`, 1167–1169, 1172–1173
- **FileUpload server control,** 164–174
  - file size limitations, 167–170
  - overview of, 164
  - permissions, 167
  - uploading file contents into `Byte` array, 173–174
  - uploading file contents into `Stream` object, 172–173
  - uploading files using, 164–167
  - uploading multiple files from same page, 170–172
- `FileWebRequest` **class,** 1188–1189
- `FileWebResponse` **class,** 1188–1189
- **FilteredTextBoxExtender control,** 959–961
- **filtering**
  - adding to LINQ query, 469–471
  - traditional query methods, 461–462
  - using `SelectParameter` controls, 294–296
- `FilterMode` **property, FilteredTextBoxExtender control,** 961
- `FilterParameters`, **DetailsView control,** 349
- `FilterType` **property, FilteredTextBoxExtender control,** 960
- `$find()` **method, Silverlight control,** 1625
- `FindControl()` **method**
  - cross-page posting, 29
  - dynamically adding user control to Web page, 1199
  - getting at server controls on master page, 244–245
- `FindNode()` **method,** `TreeView` **control,** 690, 693
- `FindProfilesByUserName()` **method,** `ProfileManager` **class,** 754
- `FindUsersInRole()` **method,** `Roles` **class,** 806–807
- `FinishButtonClick` **event, Wizard server control,** 184
- **FireBug,** 1078
- **Firebug,** 1584
- **Firebug Lite,** 1584
- **firewalls, debugging SQL,** 1134
- **first letter pseudo element, CSS,** 871
- **first-child pseudo classes, CSS,** 870
- **first-line pseudo element, CSS,** 871
- **500 errors, exception handling,** 1135
- `float` **property, CSS elements,** 882–883
- **flushing Web events,** 1492
- `Focus()` **method, TextBox server control,** 112
- **folders**
  - adding to installer output, 1556–1557
  - authenticating specific, 1016–1017
  - creating proper structure for themes, 268–269
- **folders, application,** 33–39
  - `\App_Browsers`, 39
  - `\App_Code`, 33–38
  - `\App_Data`, 38
  - `\App_GlobalResources`, 39
  - `\App_LocalResources`, 39
  - `\App_Themes`, 38
  - `\App_WebReferences`, 39
- **Font tags, HTML,** 862
- `FOR XML AUTO`, 545–549
- `FOR XML PATH`, 550–551
- `FOR XML TYPE`, 550
- `ForceCopyBuildProvider`, 45–46
- `foreach` **loop, queries,** 462
- **form elements, Wizard server control,** 184–189
- **Format menu, Visual Studio Design view,** 884–887
- **Format whole document option, configuration files,** 1452
- **FormatString property, LoginName control,** 786
- **formatting, configuration files,** 1415–1416
- `<forms>` **element,** 1007–1008
- `FormsAuthentication` **class,** 1009, 1015–1016
- `FormsAuthenticationHashPasswordForStoringInConfigFile()` **method,** 1006
- **Forms-based authentication,** 1006–1016
  - adding users to membership service, 761–775
  - configuration settings, 1428–1430
  - configuring, 1428–1430
  - defined, 997
  - `FormsAuthentication` class, 1009, 1015–1016
  - mixing versions of .NET Framework, 1570–1571
  - overview of, 1006–1110
  - Security Setup Wizard, 1502–1503
  - setting up Web site for membership, 758–761
  - using Login control with, 1014–1015
  - against values in database, 1012–1014
  - against values in `web.config` file, 1010–1012
- **FormView server control,** 360–365, 893
- **400 errors, exception handling,** 1135
- **Framework Class Library,** 1050
- **Friendly Control Adapters, CSS,** 893

- `from` **statement, LINQ syntax,** 466
- **FrontPage Extensions,** 5–6
- **FTP (File Transfer Protocol)**
  - changing application location with, 4–5
  - Copy Web Site GUI option, 1535
  - using `FtpWebRequest/FtpWebResponse`, 1186–1188
- `FtpWebRequest` **class,** 1186–1188
- `FtpWebResponse` **class,** 1186–1188
- **full trust settings, server configuration files,** 1412
- **fully qualified redirect URLs, runtime settings configuration,** 1436
- **functional areas, IIS,** 7, 558

G

- **GAC (Global Assembly Cache),** 1313–1314, 1323
- **Gacutil.exe,** 1313
- **garbage collection, .NET,** 1309
- **General section, .NET compilation,** 1518
- **GenerateMachineKey tool,** 1066
- `GeneratePassword( )` **method, creating users,** 792–793
- **generic handlers,** 1289–1293
- `GetAllProfiles( )` **method,** `ProfileManager` **class,** 754
- `GetAllRoles( )` **method,** `Roles` **class,** 801
- `GetCallBackEventReference` **arguments,** 1230
- `GetCallBackResult( )`**method, asynchronous callbacks,** 1227–1231
- `GetCallbackResults` **method,** `ICallbackEventHandler`**,** 95
- `GetCodeExpression` **method, expressions,** 371
- `GetCompletionList( )` **method, AnimationExtender control,** 944
- `GetCurrentDirectory( )` **method,** `Directory` **class,** 1147–1148
- `GetDirectories( )` **method,** `DirectoryInfo` **class,** 1144–1146
- `GetDynamicContent( )` **method, DynamicPopulateExtender control,** 957–959
- `GetFiles( )` **method,** `DirectoryInfo` **class,** 1149–1154
- `GetInvalidFileNameChars` **method,** `Path` **class,** 1155
- `GetInvalidPathChars` **method,** `Path` **class,** 1155
- `GetItem( )` **method,** `SessionStateModule`**,** 1056
- `GetItemExclusive( )` **method,** `SessionStateModule`**,** 1056
- `GetNumberOfUsersOnline( )` **method,** `Membership` **class,** 786–788
- `GetSlides( )` **method, SlideShowExtender control,** 978–979
- `GetUpdatedTime` **method, Post-Cache Substitution Control,** 1076–1077
- `GetUsersForRole ( )` **method,** `Roles` **class,** 805
- `GetUsersInRole ( )` **method,** `Roles` **class,** 605, 804
- `GetXmlDocument` **method, XmlDataSource control,** 308
- **Global Application Class,** 51–54
- **Global Assembly Cache (GAC),** 1313–1314, 1323
- **global resources, localization,** 1403–1406
- `Global.asax` **file,** 51–54
- **globalization**
  - IIS Manager, 1518
  - server-side culture declarations, 1386–1387
  - through localization. *See* localization
- **GridView control**
  - adding paging to, 325–328
  - adding sorting to, 323–325
  - binding to directory files, 1150–1153
  - creating LINQ query bound to, 465–468
  - customizing columns in, 328–331
  - DetailsView server control vs., 344
  - displaying data with, 318–321
  - overview of, 318
  - reading from event log, 1462–1464
  - in traditional query methods, 457–461
  - using `EmptyDataText` and `EmptyDataTemplate` properties, 321–323
  - using TemplateField column, 331–334
  - using with DetailsView server control, 345–349
- **GridView control, editing,** 318–344
  - deleting data, 341–343
  - errors when updating data, 337–339
  - other formatting features, 343–344
  - row data, 334–337
  - using TemplateField EditItem template, 339–341
- **Group Container, ListView,** 356–357
- **grouping data**
  - with `group` keyword in LINQ, 472–473
  - LINQ to Objects, 472–473
  - LINQ to SQL, 486–487
  - LinqDataSource control, 305–306
  - traditional query methods, 462–464
- `GroupName` **property, RadioButton control,** 135
- **groups**
  - authenticating/authorizing, 1002–1003
  - CSS selector, 872
  - personalization, 730–732, 742–743
  - rendering in ListView, 356–357
  - validation, 222–227
- **GUI tool, SQL Server Setup Wizard,** 593–594

GUIDs (unique identifiers), 243–245, 738–739
`GZipStream`, 1176–1178

H

`HandleCssClass` **property, ResizableControlExtender control,** 974
handlers
  custom configuration, 1458–1460
  custom section creation, 1453–1460
  HttpHandlers. *See* HttpHandlers
  processing HTTP requests, 1275–1278
hashed message authentication code (HMAC), 1066
header, Wizard server control, 181–182
`HeaderText` **attribute, Wizard server control,** 181–182
health monitoring
  buffering Web events, 1490–1492
  configuration, 1479–1486
  e-mailing Web events, 1492–1497
  provider model, 1477–1479
  routing events to SQL server, 1487–1490
  using Web event providers for, 612–613
  writing events via configuration, 1486–1487
HiddenField server control, 162–164, 1063, 1064–1066
hierarchical configuration, IIS Manager, 577–581
hierarchical structure, of XML data, 549
HMAC (hashed message authentication code), 1066
Home tab, Web Site Administration Tool, 1501
`HorizontalAlign` **attribute, Panel server control,** 154–156
hosting, WCF service in console application, 1367–1368
hot-keys, 109–110
hotspots, ImageMap control, 190–191
hover style, Menu control, 696
HoverMenuExtender control, 961–962
HRESULT values, 1309
HTML
  compiling, 44
  developing skill in, 72
  rendering server controls at runtime, 1210–1214
  in state management, 1034
  styling server controls, 1217–1220
  user controls vs. standard Web page, 1194
HTML and CSS design, 861–893
  caveats, 862
  external style sheets, 863–865
  Friendly Control Adapters, 893
  HTML vs. CSS, 861
  inline styles, 866
  internal style sheets, 865–866
  introducing CSS, 863
  limitations of styling HTML, 862–863
  style inheritance, 875–876
HTML and CSS design, CSS rules, 866–875
  merged styles, 872–875
  overview of, 866–867
  pseudo classes, 869–871
  pseudo elements, 871
  Selector combinations, 872
  Selector grouping, 872
  Selectors, 867–869
  styling ASP.NET controls using, 892–893
HTML and CSS design, element layout and positioning, 876–884
  controlling style overriding with `! important` attribute, 883–884
  CSS box model, 877–880
  overview of, 876
  positioning CSS elements, 880–883
HTML and CSS design, in Visual Studio, 884–893
  Apply Styles tool window, 888
  CSS Properties tool window, 888–891
  Manage Styles tool window, 887–888
  managing relative CSS links in masterpages, 891
  overview of, 884–885
  styling ASP.NET controls, 891–893
  working with CSS, 886–887
HTML elements
  applying CSS styles directly to, 72–73
  changing styles using, 72
  turning into server controls, 76–79
HTML server controls, 76–83
  HTML classes for working with, 80–81
  `HtmlContainerControl` base class, 80
  `HtmlControl` base class, 79–80
  `HtmlGenericControl` class, 81–83
  turning HTML elements into, 76–79
  Web server controls vs., 64–65, 108
`HtmlContainerControl` **class,** 80
`HtmlControl` **base class,** 79–80
HTMLDog Web site, 1590
`HtmlGenericControl` **class,** 81–83
`HtmlTextWriter` **class,** 1212–1214, 1220
`HTTP GET` **requests, state management,** 1034
HTTP Headers, 1034, 1078–1080
HTTP output, 1178–1181, 1280–1282
HTTP requests, processing
  ASP.NET request processing, 1277–1278
  IIS 5/6 and ASP.NET, 1275–1276
  overview of, 1275
  using HttpHandlers. *See* HttpHandlers
  using HttpModules. *See* HttpModules
HTTP Status Codes, 1136–1138
HTTP transmission protocol, 1003
`HttpApplication` **object,** 1036–1037

`HttpCachePolicy` **class,** 1078–1080
`HttpContext`, 1038
`HttpContext.Current.Items`, 1067–1068
`HttpFileCollection` **class,** 170–172
HTTP-GET, 1342–1344
HttpHandlers
  generic handlers, 1289–1293
  HTTP pipeline model handling requests with, 1277–1278
  mapping file extensions in IIS, 1293–1295
HttpModules, 1278–1295
  compressing HTTP output, 1178–1181
  configuration settings, 1416
  creating, 1278–1279
  HTTP pipeline model handling requests with, 1277–1278
  HttpHandlers vs., 1289
  IIS wildcards, 1286–1288
  modifying HTTP output with, 1280–1282
  performing URL rewriting with, 1283–1286
HTTP-POST, 1344–1345
`<httpRuntime>` **section,** 1436
`HttpSessionState` **object,** 1037–1038
`HttpWebRequest` **class,** 1183–1186
`HttpWebResponse` **class,** 1183–1186
HyperLink server control, 120–121
HyperLinkField control, 331
hyperlinks
  appearance of, 273–274
  attaching styles sheets in Manage Style Tool window, 887–888
  managing CSS in masterpages, 891

I

`IAsyncResult` **interface,** 434–436, 1359
`ICallbackEventHandler`, 89–90, 94–95
icons, TreeView control, 683–684
`id` **attribute,** `<browser>` **element,** 1425
`ID` **attribute, server controls,** 108, 1217
ID Selectors, CSS, 869
identification, anonymous user, 1430
`<identity>` **element,** 1024–1025
`IDisposable`, 525, 549
IE Web Developer Toolbar, Microsoft, 1586
`ie.browser` **files, configuration settings,** 1425
IEnumerable interface, 469
`If Then` **statement, CheckBox control,** 131
`IgnoreFileBuildProvider`, 45–46
IHttpHandler interface, 1291–1292
IHttpModule interface, 1277–1278
IIS (Internet Information Server)
  ASP.NET Development Server vs., 1124–1125
  authentication schemes, 759
  changing where application is saved, 3–4
  mapping file extensions, 1293–1295
  state management, 1034
  Windows authentication. *See* Windows-based authentication
  working with file extensions, 1027–1030
IIS (Internet Information Service) Manager, 569–584
  Application Pools node, 570–574
  Application Settings, 1522–1523
  changing provider for sessions using, 588–589
  configuring, 1514–1517
  Connection Strings, 1523–1524
  delegation, 581–584
  hierarchical configuration, 577–581
  .NET Compilation, 1517–1518
  .NET Globalization, 1518
  .NET Profile, 1518–1520
  .NET Roles, 1520
  .NET Trust Levels, 1520–1521
  .NET Users, 1521–1522
  overview of, 569–570
  Pages and Controls, 1524
  Providers, 1524
  Session State, 1524–1526
  Sites node, 575–576
  SMTP e-mail, 1526–1527
  summary review, 1527
  using IIS 7.0 Manager, 1032
IIS 5 (Internet Information Server, 5.0)
  adding wildcards in, 1287
  processing HTTP requests, 1275–1277
IIS 6 (Internet Information Server, 6.0)
  adding wildcards in, 1287–1288
  interaction with ASP.NET, 562–564
  main features of, 557–558
  moving to IIS 7, 584–586
  processing HTTP requests, 1275–1277
  security, 1025–1032
IIS 7 (Internet Information Server, 7.0), 557–586
  ASP.NET integrated pipeline with, 562–564
  building customized Web server, 564–569
  extensible architecture of, 562
  IIS Manager. *See* IIS (Internet Information Service) Manager
  IIS-WebServer, 558–562
  modular architecture of, 557–558
  moving application from IIS6 to, 584–586
IIS-ApplicationDevelopment update, 559
IIS-CommonHttpFeatures update, 559
IIS-FTPPublishingService update, 562
IIS-HealthAndDiagnostics update, 560
IIS-IIS6Management Compatibility update, 561
IIS-Performance update, 560
IIS-Security update, 560
IISTraceListener, 1119

- `IISWebEventProvider`**,** 613, 619
- **IIS-WebServer,** 558–562
  - IIS-ApplicationDevelopment update, 559
  - IIS-CommonHttpFeatures update, 559
  - IIS-FTPPublishingService update, 562
  - IIS-HealthAndDiagnostics update, 560
  - IIS-Performance update, 561
  - IIS-Security update, 560
  - IIS-WebServerManagement Tools update, 561
  - overview of, 558–559
  - update dependencies, 565
- **IIS-WebServerManagementTools update,** 561
- **IIS-WebServerRole,** 558
- **ILMerge,** 541
- **Image server control,** 138–139
- **ImageButton server control,** 119–120
- **ImageMap server control,** 189–191
- **images**
  - BulletedList control, 159
  - DropDownExtender control, 951–955
  - error notification, 221–222
  - Hyperlink control, 121
  - menu item, 698–700
  - ResizableControlExtender control, 993–994
  - SlideShowExtender, 977–979
  - themes, 275–278
  - TreeView control icons, 684
  - UpdateProgress control, 924–925
  - Web Part verb, 839–840
- `Images` **folder, themes,** 275–278
- `ImageUrl` **property**
  - Image server control, 138
  - ImageButton server control, 119–120
  - Web Part verbs, 839–840
- **Immediate window,** 1106
- **impersonation,** 1024–1025
- `@Implements` **directive,** 13, 21
- `@Import` **directive,** 13, 19–21, 52–54
- `import` **statement, CSS,** 865
- `Imports` **keyword, XML namespace,** 508–509
- `include` **files,** 229–231, 1435
- **indentation, VB in Visual Studio,** 518–520
- `Index` **property, File System Editor,** 1555
- **influential blogs, online resources,** 1627
- **Infrastructure Error Events Raised performance counter,** 1469
- **inheritance**
  - applying configuration files, 1414–1415
  - CSS Properties Tool window and, 890–891
  - CSS style, 875–876
- `Inherits` **attribute,** `Page` **directive,** 12
- `Init` **method, IHttpModule interface,** 1278, 1281
- `Initialize()` **method**
  - `ProviderBase` class, 634
  - `SessionStateModule`, 1056
  - `XmlMembershipProvider` class, 644–646
- `InitializeRequest()` **method,** `SessionStateModule`**,** 1056
- `InitialValue` **property, RequiredFieldValidator control,** 200–201
- **inline boxes,** 877–879
- **inline styles,** 72, 866, 892
- `InnerHtml` **property,** `HtmlContainerControl` **class,** 80
- `InnerText` **property,** `HtmlContainerControl` **class,** 80
- **in-place compilation,** 41
- `InProc` **provider,** 588
- **InProc Session model**
  - limitation of, 1039
  - overview of, 1038–1039
  - storing data in `Session` object, 1040–1043
  - Web gardening and, 1039
- **In-Process Session State**
  - maintaining state, 1058–1059
  - `Session_OnEnd` event in, 1037
  - Web gardening not used in, 1039
- `InProcSessionStateStore`
  - configuring `sessionState` management, 1038
  - defined, 610
  - overview of, 1038–1043
  - working with, 611
- `Insert` **method, cache dependencies,** 1081
- `InsertCommand`**, DetailsView control,** 350–351
- **inserting data**
  - into SQL Server, 380–381
  - using DetailsView server control, 350–351
  - using LINQ to SQL, 490–491
  - using stored procedures, 491–493
- **InsertItemTemplate, ListView control,** 359, 416
- **InsertTemplate, GridView control,** 334
- **installation**
  - AJAX Control Toolkit, 929–930
  - building programs, 1539–1547
  - using Windows Installer service. *See* Windows Installer service
- **instrumentation,** 1461–1498
  - application tracing, 1476
  - browser-based Administrative Tool building, 1470–1476
  - buffering Web events, 1490–1492
  - e-mailing Web events, 1492–1497
  - event log, 1461–1462
  - event log reading, 1462–1464
  - event log writing, 1464–1468
  - health monitoring configuration, 1479–1486
  - health monitoring provider model, 1477–1479
  - performance counter viewing through Administration Tool, 1468–1470
  - routing events to SQL server, 1487–1490
  - summary review, 1498
  - writing events via configuration, 1486–1487
- **Integrated Windows authentication,** 1004
- **IntelliSense,**
  - for CSS, 864, 866
  - for inline coding, 10
  - for LINQ to SQL, 481

IntelliSense *(continued)*
  for personalization properties, 728–729
  for server controls, 66–67
interfaces
  creating for provider Web Part, 851–854
  WCF service contract, 1365–1367
  XML Web service, 1333–1336
internal style sheets, CSS
  combining with external style sheets, 876
  creating, 73–75
  overview of, 865–866
internationalization, through localization. *See* localization
Internet Explorer
  client-side culture declarations, 1387–1389
  Design mode only in, 825
  viewing or editing XAML using, 1614–1617
Internet Information Server. *See* IIS (Internet Information Server)
Internet Information Service Manager. *See* IIS (Internet Information Service) Manager
Interop Type Library, 1314–1316
`InvalidChars` **property, FilteredTextBoxExtender control,** 961
invariant cultures, 1382–1383, 1400–1401
IP addresses, restrictions, 1025–1027
IPostBackDataHandler interface, 1242–1244
IPostbackEventHandler, 89
ISAPI Rewrite module, Helicon, 1599
`IsApplication` **property, File System Editor,** 1555
`IsAuthenticated` **property,** `User.Identity`, 1018–1019
`IsCrossPagePostBack` **property, cross-page posting,** 31–33
`IsEnabled` **property, page-level tracing,** 1108
`IsLockedOut` **property,** `MembershipUser` **object,** 781, 783–784
`IsPostBack` **property,** `Page` **class,** 26
`IsReusable` **property, IHttpHandler interface,** 1291
`IsUserInRole ( )` **method,** `Roles` **class,** 605, 806
Item Container, ListView, 356–357
ItemTemplate
  DataList control, 404–407
  GridView control, 333
  ListView control, 414–415
`IXPathNavigable`**,** `XslCompiledTransform` **class,** 538

J

JavaScript. *See also* AJAX (Asynchronous JavaScript and XML)
  accessing Silverlight elements from events in, 1625–1626
  AJAX client-side technologies, 900–901
  AJAX dependency on, 898–899
  Aptana Studio tools for, 1588
  buttons for client-side, 117–118
  client-side callback functions, 1231
  client-side debugging, 1131–1134
  compressor, 1593–1594
  hard-coded .js files and ASP.NET 3.5, 1573–1574
  hooking up events to Silverlight, 1616–1617
  libraries, 914
  manipulating server controls, 83–88
  receiving Silverlight events in, 1623–1625
  resource for, 89
  in Web applications, 261
JavaScript Object Notation (JSON) editor, 1588
JIT (just-in-time) compiler, 1122–1124
joins
  LINQ to Objects, 473–475
  LINQ to XML, 478–479
`.js` **(JavaScript) file extension,** 1573–1574
Just My Code debugging, 1130–1131
just-in-time (JIT) compiler, 1122–1124

K

`keyContainerName` **attribute,** `RsaProtectedConfigurationProvider`**,** 623
`keyEntropy` **attribute,** `DpapiProtectedConfigurationProvider`**,** 622
keywords, LINQ query syntax, 466–469

L

Label server control
  FileUpload control using, 166–167
  Literal control vs., 110
  overview of, 108–110
  using with DragPanelExtender control, 951
`Label2Answer` **key, resource files,** 1400–1401
`LabelStartText` **property,** `WebPart` **class,** 848, 850
language pseudo classes, CSS, 871
languages
  cultures and regions, 1381–1382
  resource files, 1400–1401
  setting preferences, 1387–1389
late-bound access, personalization, 728–730
Launch Conditions Editor, 1543, 1568–1569
layout
  DataList control template, 403–410
  menu item, 698
  Web Part settings, 833
  Web zone, 814–817
  WebPartZone control, 819
layout, CSS elements, 876–884
  controlling style overriding, 883–884
  CSS box model, 877–880

  positioning, 880–883
  properties, 876
LayoutTemplate, ListView control, 352–354, 412–414
`Legacy` **setting, XHTML,** 1572
limited object serialization formatter (LosFormatter), 1064
`LimitedSqlRoleProvider` **provider,** 651–652, 656–660
line numbers, and debugging, 1123
`LineImagesFolder` **property, TreeView control,** 686–687
link pseudo classes, CSS, 870
`link` **tags, HTML,** 864–865
LinkButton server control, 119, 958
links. ***See* hyperlinks**
LINQ (Language Integrated Query)
  extending, 494–495
  LINQ to Objects. *See* LINQ to Objects
  LINQ to SQL. *See* LINQ to SQL
  LINQ to XML. *See* LINQ to XML (XLINQ)
  summary review, 495
  traditional query methods, 455–464
LINQ to Objects, 455–476
  adding query filters, 468–472
  defined, 455
  delayed execution behavior of, 468
  grouping data, 472–473
  joins, 473–475
  operators used by, 473
  paging, 475–476
  replacing traditional queries, 464–468
  traditional query methods, 455–464
LINQ to SQL, 481–494
  adding new data context, 483–484
  deleting data, 493–494
  inserting data, 490–493
  overview of, 481
  setting up file, 482–483
  updating data, 493
  writing LINQ queries, 484–490
LINQ to XML (XLINQ)
  bridging `XmlSerializer` and, 523–524
  creating CLR objects from XML, 518–519
  creating XML, 522–524
  joining XML data, 479–481
  overview of, 497–498
  using `XDocument` vs. `XmlReader`, 508–509
  using `XPath` with `XDocuments`, 529
  validating against schema with `XDocument`, 511–513
  working with, 476–479
LINQ to XSD, 518
LinqDataSource Configuration Wizard, 302–304
LinqDataSource control, 302–307
  data concurrency, 306–307
  events, 307
  overview of, 302–304
  query operations, 304–306
ListBox server control, 125–129
  adding items to collection, 129
  allowing users to select multiple items, 126
  as databound control, 365
  example, 126–129
  overview of, 125–126
  using ListSearchExtender control, 962–964
ListSearchExtender control, 962–964
ListView server control, 350–360, 410–419
  data binding and commands, 357–359
  data item rendering, 355–356
  getting started with, 352–354
  group rendering, 356–357
  overview of, 350–352
  paging and DataPager control, 359–360
  results of, 417–419
ListView server control, templates
  creating EditItemTemplate, 415
  creating EmptyItemTemplate, 415–416
  creating InsertItemTemplate, 416
  creating ItemTemplate, 414–415
  creating LayoutTemplate, 412–414
  data item rendering using, 355–356
  defining, 352–353
  group rendering using, 356–357
  list of available, 410–411
  overview of, 354–355
  using, 411–412
  using EmptyItem template, 357
Literal server control, 110–111
`Load` **event,** 25
`Load` **method,** `XslCompiledTransform` **class,** 538–541
`LoadControl()` **method, to Web page,** 1198–1203
`LoadPostData()` **method, IPostBackDataHandler interface,** 1242–1244
Local IIS setting, Copy Web Site GUI, 1535
local resource files, 1397–1400
localization, 1381–1407
  ASP.NET threads, 1383–1386
  client-side culture declarations, 1387–1389
  culture types, 1382–1383
  date translation, 1389–1391
  `Default.aspx` page finalization, 1401–1403
  DLLs, AJAX controls, 936
  global resources, 1403–1406
  language resource files, 1400–1401
  local resource files, 1397–1400
  neutral cultures, 1403
  number and currency translation, 1391–1394
  overview of, 1381–1382
  resource editor, 1406–1407
  server-side culture declarations, 1386–1387
  site map, 710–714
  sorting strings translation, 1394–1397
`Localization` **property, Windows Installer,** 1553
`LocalName` **property,** `Reader`**,** 508
`LocalSqlServer`**,** `SqlMembershipProvider`**,** 600
location
  authorization configuration, 1431

- **location** ***(continued)***
  - file authorization, 1432
  - using built-in Web server, 2–3
  - using FrontPage Extensions for remote sites, 5–6
  - using FTP, 4–5
  - using IIS, 3–4
- `Lock()` **method,** `Application`**,** 1059
- **lock statement,** `ReadUserFile()` **method,** 650
- **lock-down configuration settings,** 1433
- `Log` **command,** 1106
- `<LoggedInTemplate>`**, LoginView control,** 793–794
- **Logging buffer mode,** 1492
- **login,** 776–784
  - denying unauthenticated users, 776
  - locking out users with bad passwords, 780–784
  - logging in users programmatically, 779–780
  - using Login server control, 776–779
  - using `XmlMembershipProvider`, 651–652
- **Login server control,** 776, 1014–1015
- **LoginName server control,** 784–786
- **LoginStatus server control,** 784
- `loginURL` **attribute, Forms authentication,** 1429
- **LoginView server control,** 793–795
- `LogVisits` **property, File System Editor,** 1556
- **LosFormatter (limited object serialization formatter),** 1064

M

- `m` **variable, LINQ syntax,** 466
- `machine.config` **file**
  - adding SQL data provider, 745–746
  - changing SQL Server 2005 connection string, 595–598
  - configuration files. *See* configuration
  - configuring providers, 625
  - controlling ASP.NET cache, 1081
  - creating `SqlMembershipProvider`, 598–600
  - creating `SqlRoleProvider`, 602–604
  - defined, 996
  - membership provider settings, 774–775
  - using `RsaProtectedConfigurationProvider`, 622–623
  - using SQL Server as provider, 748
- `<machinekey>`, `web.config` **file,** 1066
- `<MailDefinition>` **element, PasswordRecovery control,** 790
- **MaMaskedEditValidator control,** 964–966
- **Manage Styles tool window, Visual Studio,** 887–888
- **management**
  - with IIS Manager. *See* IIS (Internet Information Service) Manager
  - with Web Site Administration Tool. *See* Web Site Administration Tool
- **Management Objects, ASP.NET,** 1441–1448
- `Manufacturer` **property, Windows Installer,** 1553
- `ManufacturerUrl` **property, Windows Installer,** 1553
- **mapping**
  - file extension in IIS, 1293–1295
  - file extensions, 1027–1030
  - with File Types Editor, 1559–1561
  - IIS wildcard, 1287–1288
  - LINQ to XML, 478–479
  - URLs, 709–710
- **MARS (Multiple Active Result Sets)**
  - asynchronous command execution using, 432
  - defined, 454
  - using multiple wait handles, 442–451
- **MaskedEditExtender control,** 964–966
- `@Master` **directive,** 13, 17–18
- `.master` **file extension,** 231–235
- **master pages,** 229–263
  - ASP.NET AJAX and, 259–262
  - basics, 231–233
  - caching, 259
  - coding, 233–235
  - coding content pages, 235–239
  - connecting to Web Parts, 858–860
  - container-specific, 257–258
  - development of, 229–231
  - event ordering, 258–259
  - managing CSS links, 891
  - mixing page types and languages, 239–241
  - nesting, 253–256
  - placing WebPartManager control on, 814
  - programmatically assigning, 251–253
  - specifying default content in, 250–251
  - specifying which one to use, 241–242
  - working with controls and properties from, 243–249
  - working with page title, 242–243
  - working with ScriptManagerProxy control, 914–916
- `MasterPageFile` **attribute, content page,** 237–241, 257–258, 1434
- `@MasterType` **directive,** 13, 23
- `maxBatchGeneratedFileSize` **attribute, compilation configuration,** 1422
- `maxBatchSize` **attribute, compilation configuration,** 1422
- `MaximumValue` **property, RangeValidator control,** 206–209
- `maxInvalidPasswordAttempts` **attribute,** 781–783
- `maxRequestLength` **property**
  - changing file-size limitation setting, 169–170
  - runtime settings configuration, 1437

`.mdf` **files. *See* SQL Server Express Edition (`.mdf` file)**
medium trust setting, 1412–1413
Membership API, 770–772, 809
membership management service
  adding users, 761–775
  asking for credentials, 776–784
  authorization vs. authentication, 758
  dealing with passwords, 788–793
  overview of, 757
  public methods of Membership API, 809
  role management service vs., 799
  setting up Web site for, 758–761
  showing number of users online, 786–788
  using Web Site Administration Tool, 808–809
  working with authenticated users, 784–786
membership providers, 598–602
  changing password requirements, 764–765
  in `machine.config file`, 774–775
  .NET Users, 1521–1522
  overview of, 590–591
  setting up Web site for, 758–761
  `System.Web.Security.ActiveDirectoryMembershipProvider`, 600–602
  `System.Web.Security.SqlMembershipProvider`, 598–600
  using multiple skin options, 598
membership system, role management, 715–716
`MembershipCreateUserException`, 772–774
`MembershipProvider` **class**
  building own providers, 635
  constructing class skeleton, 636–640
  creating CustomProviders application, 635–636
  creating XML user data store, 640
  defining provider instance in web.config file, 641–642
  implementing methods/properties of, 643–646
  not implementing methods/properties of, 642–643
`MembershipUser` **object, user login,** 783–784
memory, 1074, 1309
`MemoryStream`**, writing to,** 1170
Menu server control, 693–702
  binding to XML file, 701–702
  as databound control, 366
  events, 700–701
  menu items, layout, 698
  menu items, separating with images, 699–700
  overview of, 693–694
  pop-out symbol, 698–699
  styles, adding to dynamic items, 697
  styles, for static items, 695–696
  styles, predefined, 694–695
  styles, with CSS-Friendly Control Adapters, 893
`MenuItemClick` **event,** 700–701
Merge Module Project, 1541
merge tools, 1601
merged styles, CSS, 872
`meta:resourcekey` **attribute, local resource files,** 1399
methods
  `Directory` class, 1147–1148
  extending session state with other providers, 1056
  `FormsAuthentication` class, 1015–1016
  Membership API public, 809
  overloading Web, 1346–1349
  `Path` class, 1154–1155
  `ProfileManager` class, 750–751
  Roles API public, 810
  `WebPartManager` class, 841
Microsoft programs. *See* by individual names
migrating older ASP.NET projects, 1567–1582
  converting ASP.NET 1.x in VS 2008, 1574–1579
  Forms authentication for mixing versions, 1570–1571
  migrating from 2.0 to 3.5, 1580–1582
  no hard-coded .js files in 3.5, 1573–1574
  reserved folders, 1571
  running multiple versions side-by-side, 1568
  upgrading applications, 1568–1570
  v. 3.5 pages as XHTML, 1571–1573
`MinimumValue` **property, RangeValidator control,** 206–209
`minRequiredNonalphanumericCharacters` **attribute, passwords,** 630–632, 765
`minRequiredPasswordLength` **attribute, passwords,** 632, 765
`mod_rewrite`, **Apache Web Server,** 1599
ModalPopUpExtender control, 948–949
`mode` **attribute**
  <authentication> node, 997–998
  `<sessionState>` configuration, 1418
  custom errors configuration settings, 1427
  Literal server control, 110–111
modes, Web page
  changing. *See* WebPartManager control
  modifying Web Part settings, 828–833
  possible, 812
  session state settings, 1525–1526
  simpler password structures with, 630–631
modules
  architecture of IIS 7, 557–558
  HttpModules. *See* HttpModules
money, currency translation, 1392–1394
monitoring, health. *See* health monitoring
`mostRecent` **property, trace data,** 1112

`.msi` **file extension, Windows Installer service,** 1540
MSMQ, 1360
multiline text boxes, 112
Multiple Active Result Sets. ***See*** **MARS (Multiple Active Result Sets)**
MultiView server control, 174–178
MutuallyExclusiveCheckBox Extender control, 967–968
MYKEY private variable, 1045
MyKey property, 1043–1045
`myRead.Read( )` **method,** `DataReader` **class,** 379

N

`name` **attribute**
  `<add>` element, 1093
  `<profiles>` configuration, 1486
  `<rules>` configuration, 1484
  Forms authentication, 1429
`Name` **property,** `ProviderBase` **class,** 633
`Name` **property,** `Reader`, 508
namespaces
  `<pages>` configuration, 1435
  ADO.NET, 383–384
  LINQ, 465
namespaces, XML
  building Web services, 1379
  declaring, 508–509
  defined, 499
  overcoming `XmlDataSource` control limitations, 533–534
  resolution, 528–529
`NameTable` **optimization, XML,** 513–515
`NameValueFileSectionHandler`, 1453–1456
naming conventions, LINQ to SQL, 482–483, 495
`NavigateUrl` **attribute, ImageMap control,** 191
`NavigateUrl` **property,** `TreeNode` **object,** 690–693
navigation, Wizard control, 182–183
nesting
  master pages, 253–256
  `.sitemap` files, 720–722
  `web.config` files, 996
.NET
  Compilation, 1517–1518
  deploying COM components with, 1312–1313
  Globalization, 1518
  Profile, 1518–1520
  Remoting, 1360
  Roles, 1520
  running multiple Web sites with multiple versions of, 1439
  server configuration files, 1411–1413
  Trust Levels, 1520–1521
  Users, 1521–1522
.NET from unmanaged code, 1314–1323
  COM-Callable Wrapper, 1314–1316
  deploying .NET components with COM apps, 1322–1323
  early vs. late binding, 1320
  error handling, 1320–1322
  overview of, 1314
  using .NET components within COM objects, 1316–1319
`net start` **command,** 1419
.NET Users, IIS, 1521–1522
network communications, 1182–1190
  defined, 1182
  `FileWebRequest` and `FileWebResponse`, 1188–1189
  `FtpWebRequest` and `FtpWebResponse`, 1186–1188
  `HttpWebRequest` and `HttpWebResponse`, 1183–1186
  sending mail, 1189–1190
`NetworkStream`**, reading to,** 1170–1171
neutral cultures, 1382–1383, 1403
New Group dialog, authentication/authorization, 1003
New Project dialog, 1204, 1298
New Style dialog box, Visual Studio, 886–887
New User dialog, Windows authentication, 998–999
New Web Site dialog, 902–903, 1327–1328
`NewPasswordRegularExpression` **attribute, ChangePassword control,** 789
`NextView` **method, MultiView and View controls,** 176
NoBot control, AJAX, 988–990
nodes, TreeView control
  adding programmatically, 690–693
  expanding and collapsing programmatically, 687–690
  overview of, 676
  specifying lines used to connect, 685–687
None, authentication, 997, 1429
Normal Mode, Portal Framework Web pages, 812
Normal positioning, CSS elements, 880–881
Northwind.mdf file, locating, 378
Notepad, editing XAML in, 1620
NT authentication, SQL Server, 2005, 1134
NTLM, 1004
`Null` **value, GridView control,** 328
numbers, localizing, 1391–1394
NumericUpDownExtender control, 968–969
`numRecompilesBeforeAppRestart` **attribute, compilation,** 1423

O

OAEP (Optional Asymmetric Encryption and Padding), 623
Object Relation (O/R) mapper, LINQ to XML, 481
Object Test Bench, 55, 60–61
ObjectDataSource control, 309–313
ODP (Oracle Data Provider), 299
offline, taking application, 1512
`OnClientClick` **attribute, Button control,** 118
`OnCommand` **event, Button control,** 116
`OnDataBound` **attribute, TreeView control,** 689

- `OnGenerateChallengeandResponse` **property, NoBot control,** 989
- **online, displaying number of users,** 786–788
- **online resources**
  - advanced Web services, 1346
  - AJAX Control Toolkit download, 929–930
  - ASP.NET 3.5 Extensions Preview, 1621
  - ASP.NET ultimate, 1590–1591
  - author blogs, 1627
  - Cassini built-in Web server, 2, 1124
  - DbgView, 1113
  - extending LINQ, 493–494
  - Fiddler tool, 671
  - Friendly Control Adapters toolkit, CSS, 893
  - GenerateMachineKey tool, 1066
  - HTTP Headers and controlling caching, 1078
  - IIS 7 and ASP.NET integration, 1277
  - ILMerge, 541
  - influential blogs, 1627
  - JavaScript for controls, 84
  - LINQ to XSD, 518
  - MSDN URL for debugging SQL Server, 1134
  - Northwind database, 378
  - regular expression strings, 211
  - third-party session-state providers, 1057
  - third-party validation server controls, 196
  - Ultimate Tools List updates, 1583
  - ViewState, 1066
  - ViewStateDecoder tool, 1066
  - W3C on XML, 500
  - Web sites, 1628
  - working with HTML, 72
  - working with JavaScript, 89
  - XML Schema Editor, 505
  - XSLT, 538
- `OnServerValidate` **attribute, CustomValidator control,** 216
- `OnUpdating` **event, UpdatePanelAnimationExtender control,** 983–984
- `Opacity` **property, DropDownExtender control,** 955
- `OpenMachineConfiguration` **method,** `machine.config`, 1446–1447
- `Operator` **property, CompareValidator control,** 205
- **operators, LINQ query,** 473
- **Optimistic Currency, SqlDataSource control,** 293
- **Optional Asymmetric Encryption and Padding (OAEP),** 623
- **O/R (Object Relation) mapper, LINQ to XML,** 481
- **Oracle Data Provider (ODP),** 299
- **Oracle database,** 299–301, 400–403
- `order by` **statement, LINQ query,** 467–468
- `OrderBy` **clause, LinqDataSource control,** 305
- `Orientation` **attribute, Menu control,** 698
- **Out-of-Process Session State**
  - configuration settings, 1418–1421
  - configuring `sessionState` management, 1038
  - maintaining state using, 1058–1059
  - working with, 1046–1051
- `OutOfProcSessionStateStore`, 610–612
- **output, installer,** 1556–1557
- **output caching**
  - overview of, 1071–1072
  - runtime settings configuration, 1438
  - testing SQL Server cache invalidation, 1094–1101
  - `VaryByControl` attribute, 1073
  - `VaryByCustom` attribute, 1073–1074
  - `VaryByParam` attribute, 1072–1073
- **Output File Name,** 1550–1551
- `@OutputCache` **directive,** 13, 23, 259
- **OutputCacheModule,** 1072
- `OutputDebugString`, 1113
- **overloading WebMethods,** 1346–1349
- **override rules, configuration files,** 1415

P

- **package files,** 1551
- **packaging.** ***See*** **application packaging and deploying**
- **Packer for .NET,** 1593–1594
- **page, cross posting,** 27–33
- `Page` **class**
  - converting ASP.NET 1.x in VS 2008, 1578
  - `Error` property, 1135
  - making sessions transparent, 1041–1043
  - setting and retrieving objects from session, 1050–1051
  - `Trace` property, 1108
- **page compilation,** 40–44
- `@Page` **directive**
  - adding tracing to, 1108
  - converting ASP.NET 1.x in Visual Studio 2008, 1578
  - defined, 13
  - optimizing `Session` performance, 1045–1046
  - overview of, 14–17
  - server-side culture declarations, 1387
- **page directives,** 13–24
  - `@Assembly` directive, 22
  - `@Control` directive, 18–19
  - `@Implements` directive, 21
  - `@Import` directive, 19–21
  - `@Master` directive, 17–18
  - `@MasterType` directive, 23
  - `@OutputCache` directive, 23
  - `@PreviousPageType` directive, 22
  - `@Reference` directive, 14, 24
  - `@Register` directive, 14, 21–22
- **page events,** 24–26
- **page exception handling,** 1135–1136
- **page postbacks,** 26, 1238
- **page recycling,** 1130
- **page structure options, ASP.NET**
  - code-behind model, 10–12
  - inline coding, 6–10
  - overview of, 6–7
- **page tracing,** 1108

`Page_Init` **event,** 825
`Page_Load` **event**
  disabling client-side validation, 221
  DropDownExtender control, 953
  example of, 68
  getting at server controls on master page, 244–245
  working with simple callback, 95
`Page_LoadComplete` **event handler, controls on master page,** 244–245
`pageBaseType` **element,** `<pages>` **configuration,** 1434
PageCatalogPart control, 826, 836
`PageClientScript` **property, applying JavaScript to controls,** 84
`PageIndexingChanged` **event, GridView control,** 327
`PageIndexingChanging` **event, GridView control,** 327
`Page.MasterPageFile` **property,** 251–253
`Page.RegisterClientScriptBlock` **method,** 84
`Page.RegisterStartupScript` **method,** 84
`PagerSettings` **property, GridView control,** 326–327
`PagerStyle` **property, GridView control,** 326–327
pages, applying themes to Web. ***See*** **themes**
Pages and Controls, IIS Manager, 1524
`<pages>`**, configuration settings,** 1433–1435
Page.Trace, 1108–1109
paging
  DataPager control, 359–360
  DetailsView control, 345
  GridView control, 325–328
  LINQ, 475–476
PagingBulletedListExtender control, 969–970
Panel server control
  adding CollapsiblePanelExtender control, 946–947
  adding DragPanelExtender control, 951
  adding DropDownExtender control with, 954–955
  adding DynamicPopulateExtender control, 956–959
  adding ResizableControlExtender control, 972–974
  adding RoundedCornersExtender control, 975–976
  adding UpdatePanelAnimationExtender control, 983–984
  overview of, 153–156
parameters
  client-side callback, 96–99
  configuring SQL statements, 392–395
  XCOPY, 1532–1533
parent node, TreeView control, 676
`parentID` **attribute, browsers,** 1425
`ParentLevelsDisplayed` **property, SiteMapPath control,** 669
`ParseExpression` **method, expressions,** 372–373
`Partial` **keyword, code-behind pages,** 12
partial page caching, 1074–1075
Passport authentication
  configuration settings, 1428
  defined, 997
  limitations of, 759
  overview of, 1016
PasswordRecovery server control, 789–792
passwords
  adding user to membership service, 761–763
  authenticating against values in database, 1012–1014
  authenticating against values in `web.config` file, 1010–1012
  ChangePassword server control, 788–789
  generating random, 792–793
  inputting on form with TextBox control, 111
  limitations of Basic authentication, 1004
  locking out users with bad, 780–784
  PasswordRecovery server control, 789–792
  protecting configuration settings, 1448
  seeing and modifying, 763–765
  using `SqlMembershipProvider`, 629–632
PasswordStrength control, AJAX, 990–991
`passwordStrengthRegularExpression` **attribute,** 632–633
"Paste XML as XLinq" feature, 523
`path` **attribute,** 1429, 1433
`PathDirection` **property, SiteMapPath control,** 668
paths
  changing in IIS Manager, 1517
  `Path` class, 1154–1158
  structuring pages with, 6
  working with, 1154–1158
`PathSeparator` **property, SiteMapPath control,** 666–668
PDB (program database or debug) file, 1122–1123
`-pe` **command,** 1449
performance, 1045–1046. ***See also*** **caching**
performance counters, 1468–1476
permissions, uploading files, 167
`Personalizable` **attribute, custom Web Parts,** 848
personalization, 723–756
  model for, 723–724
  past ways of providing end user, 723
  programmatic access to, 741–745
  providers, 745–749
personalization, anonymous, 735–741
  defined, 735
  enabling identification of end user, 736–739
  migration to authenticated users, 741–745
  options for personalization properties, 739–740
  user profile storage warnings, 740–741
  working with, 739
personalization, managing application profiles, 749–755
  overview of, 749

  - `ProfileManager` class methods, 750–751
  - `ProfileManager` class properties, 750
  - `ProfileManager.aspx` page, building, 751–754
  - `ProfileManager.aspx` page, examining code of, 754–755
  - `ProfileManager.aspx` page, running, 755–756
- **personalization properties,** 725–735
  - anonymous options for, 739–740
  - creating, 725–726
  - making read-only, 735
  - migrating anonymous users for particular, 742–743
  - providing default values, 735
  - registering users in membership service with, 766–770
  - storing in groups, 730–732
  - using, 726–730
  - using custom types, 732–735
- **personalization provider**
  - .NET Profile, 1518–1520
  - overview of, 606–608
- **Personalization Services layer, personalization model,** 724
- `PersonalizationScope`**, custom Web Parts,** 848
- `pkmgr.exe`**,** 567–568
- **Placeholder server control,** 156
- `PlayInterval` **property, SlideShowExtender control,** 979
- **Poll approach, asynchronous commands,** 436–439
- `pollTime` **attribute,** `<add>` **element,** 1092–1093
- **polymorphism, and method overloading,** 1346
- **pop-out symbol, Menu control,** 698–699
- **PopUpControlExtender control,** 970–972
- `PopUpControlID` **property, ModalPopUpExtender control,** 949
- **Portal Framework**
  - overview of, 811–813
  - Web Parts. *See* Web Parts
  - working with classes in, 841–844
- **ports**
  - session state service, 1047
  - Web services using port 80, 1341
  - working with serial, 1181–1182
- `position` **attribute, CSS box model,** 879
- **positioning CSS elements**
  - Absolute, 882
  - Normal, 880–881
  - overview of, 880
  - Relative, 881–882
  - using `float` property, 882–883
- **PositionIsEverything Web site,** 1590
- **postbacks**
  - accessing data in server controls, 1242–1244
  - client-side vs. typical, 89
  - dealing with, 26
  - handling events in server controls, 1238–1242
  - state management and, 1061–1063
  - tracking user controls across, 1202–1203
  - using Timer control for asynchronous, 916–917
  - using triggers for asynchronous, 918–921
- **PostBackTrigger,** 920
- `PostBackUrl` **attribute, Button control,** 118, 1061
- `PostBackValue` **attribute, ImageMap control,** 190–191
- `PostBuildEvent` **property, Windows Installer,** 1553
- **Post-Cache Substitution Control,** 1075–1077
- **Power Toys Pack Installer,** 1596
- `PreBuildEvent` **property, Windows Installer,** 1553
- **precompilation**
  - for business objects, 1298–1302
  - for deployment, 41–42, 1537–1539
- **predefined styles, Menu control,** 694–695
- **pre-existing providers.** ***See*** **providers, extending pre-existing**
- `PreInit` **event, assigning master pages,** 251–253
- `PreRender( )` **method, server controls,** 1224–1227
- **presentation logic, inline coding and,** 10
- `@PreviousPageType` **directive,** 13, 22
- **private assemblies,** 1312–1313, 1322–1323
- **privileges, authorization configuration,** 1430–1432
- **Process Explorer,** 1604
- `<processModel>`**element, configuration,** 1438–1439
- **processor affinity,** 1039
- `ProcessRequest( )` **method, IHttpHandler interface,** 1291–1292
- `ProductCode` **property, Windows Installer,** 1554
- `ProductName` **property, Windows Installer,** 1554
- **products, Windows Installer service,** 1540
- ***Professional IIS 7 and ASP.NET Integrated Programming*** **(Khosravi),** 1277
- **Profile, .NET,** 1518–1520
- **Profile API layer, personalization model,** 724
- `Profile` **class, personalization properties,** 728–729
- `Profile_MigrateAnonymous` **event,** 741–742
- `Profile_Personalization` **event,** 743–744
- `<profile>` **section,** `web.config` **file,** 726
- **ProfileCommon object,** 768
- `ProfileManager` **class,** 749–756
  - methods, 750–751
  - overview of, 749
  - `ProfileManager.aspx` page, building, 751–754
  - `ProfileManager.aspx` page, examining code, 754–755
  - `ProfileManager.aspx` page, running, 755–756
  - properties, 750
- `ProfileModule` **class,** 741–745
- `Profile.ProfileAutoSaving` **event,** 744–745
- **profiler tools,** 1589
- **profiles**
  - health monitoring configuration, 1485–1486

profiles *(continued)*
  managing application. *See* `ProfileManager` class
  programmatic access to personalization, 741–745
program database or debug (PDB) file, 1122–1123
programming
  authorization. *See* authorization, programmatic
  configuration files, 1441–1448
  mixing languages on content/master pages, 239–241
`<ProgressTemplate>` **element, UpdateProgress control,** 923–925
properties
  Accordion control, 987–988
  CalendarExtender control, 946
  CheckBoxList control, 133–134
  composite controls, 1245–1247
  cross-page posting, 29–33
  CSS Properties Tool window, 888–891
  custom controls, 283–285, 1207
  custom Web Part control, 848
  dragging and dropping controls, 65–66
  DropDownExtender control, 955
  file and directory, 1158–1160
  File System Editor, 1555
  File Types Editor, 1560–1561
  `FormsAuthentication` class, 1015–1016
  GridView control style, 343–344
  HoverMenuExtender control, 962
  HTML server control styles, 1217–1220
  `HtmlContainerControl` class, 80
  `HtmlControl` base class, 79–80
  ListSearchExtender control, 964
  Login control, 779
  MaskedEditExtender control, 966
  master page, 246–249
  NoBot control, 989
  NumericUpDownExtender control, 968
  PasswordStrength control, 991
  personalization. *See* personalization properties
  `ProfileManager` class, 750
  RequiredFieldValidator control, 198–202
  server controls, 70–72, 1210–1211
  SliderExtender control, 977
  SlideShowExtender control, 978–979
  styling ASP.NET controls, 891–893
  ToggleButtonExtender control, 983
  user controls, 1196–1198
  `WebMethod` attribute, 1332–1333
  `WebPart` class, 843–844
  `WebPartManager` class, 841
  `WebPartZone` class, 842–843
  `WebService` attribute, 1332
  Windows Installer, 1550–1554
PropertyGridEditorPart, 833
protection, configuration settings, 1429, 1448–1452
protocols, Web services transport, 1341–1346
provider model, 587–626
  in ASP.NET 3.5, 589–591
  configuration providers, 620–623
  configuring providers, 625–626
  health monitoring, 1477–1479
  membership providers, 598–602
  overview of, 587–588
  personalization provider, 606–608
  role providers, 602–606
  SessionState providers, 609–612
  setting up for SQL Server versions, 591–598
  SiteMap provider, 608–609
  understanding providers, 588–589
  Web event providers. *See* Web event providers
  WebParts provider, 623–625
provider model, extending, 627–660
  building providers. *See* providers, building
  modifying with attributes, 628–633
  as one tier in larger architecture, 627–628
  pre-existing providers. *See* providers, extending pre-existing
  `ProviderBase` class, 633–635
Provider pattern, `HttpSessionState` **object,** 1038
`Provider` **property,** `SiteMap` **class,** 706
Provider tab, Web Site Administration Tool, 1512–1514
`ProviderBase` **class,** 633–635
providers
  `<providers>` configuration, 1482–1483
  `<rules>` configuration, 1483–1485
  custom, 1421
  e-mailing Web events, 1493–1497
  extending session state with, 1056–1057
  health monitoring configuration, 1482–1483
  IIS Manager, 1524
  personalization, 724, 745–749
  SqlDataSource control connections, 290
  understanding, 588–589
  Web Part, 850–854, 856–858
providers, building, 635–652
  building `ReadUserFile()` method, 648–651
  constructing class skeleton, 636–640
  creating CustomProviders application, 635–636
  creating XML user data store, 640
  defining provider instance in `web.config` file, 641–642
  `MembershipProvider` class, 642–646
  user login, 651–652
  validating users, 647–648
providers, extending pre-existing, 652–660
  `AddUsersToRole()` method, 655–656

  - `CreateRole()` method, 653–655
  - `DeleteRole()` method, 655
  - limiting role capabilities, 652–653
  - using `LimitedSqlRoleProvider` provider, 656–660
- `Providers` **property,** `SiteMap` **class,** 706
- **ProxyWebPartManager control,** 859–860
- **pseudo classes, CSS,** 869–871
- **pseudo elements, CSS,** 871
- **pseudo** `web.config` **files,** 1415–1416
- **public assemblies,** 1313–1314, 1323
- **Publish Web Site,** 1538–1539

Q

- **queries, 455–464.** ***See also*** **LINQ (Language Integrated Query)**
- **QueryString (URL),** 1036, 1060
- **queues, buffering Web events,** 1490–1492
- **queues, runtime settings configuration,** 1438
- **QuirksMode Web site,** 1590

R

- **RadioButton server control,** 134–136
- **RadioButtonList server control**
  - as databound control, 365
  - DropDownList control vs., 121
  - overview of, 136–137
  - RadioButton control vs., 136
  - visually removing items from collections, 124–125
- `Radius` **property, DropDownExtender control,** 955
- `Radius` **property, RoundedCornersExtender control,** 975–976
- `RaiseCallBackEvent()` **method, asynchronous callbacks,** 1227–1231
- `RaiseCallbackEvent` **method,** `ICallbackEventHandler`**,** 95, 104
- `RaisePostBackDataChangedEvent()` **method, IPostBackDataHandler interface,** 1242–1244
- **random passwords,** 792–793
- **RangeValidator server control,** 196, 206–209
- **Rating control, AJAX,** 991–992
- **RCW (Runtime Callable Wrapper)**
  - adding reference to COM control manually, 1307
  - Com-Callable Wrapper vs., 1314
  - defined, 1302–1303
  - deploying COM with public assemblies, 1313–1314
  - error handling in .NET, 1309
  - overview of, 1303–1304
- `Read()` **method,** `Stream` **class,** 1168, 1170–1173
- `ReadContentAs()` **method,** `XmlReader`**,** 515–516
- `ReadElementContentAs()` **method,** `XmlReader`**,** 515–516
- `ReadElementString()` **method,** `XmlReader`**,** 515
- `Reader` **class**
  - encodings, 1174–1175
  - I/O shortcuts, 1175–1176
  - overview of, 1171–1174
  - working with Writers and Streams, 1166
- **reading, with event log,** 1462–1464
- **read-only, personalization,** 735
- `readOnly` **attribute, personalization properties,** 735
- `ReadSubTree`**,** 516–517, 524
- `ReadToDescendant`**,** `XmlReader`**,** 525
- `ReadToNextSibling`**,** `XmlReader`**,** 525
- `ReadUserFile()` **method,** 648–651
- `ReadWriteControlDesigner` **class,** 1259
- `RedirectFromLoginPage()` **method,** `FormsAuthentication`**,** 1009–1011, 1014
- `redirectUrl` **attribute, authentication,** 1428
- **Refactor! for ASP.NET, Devexpress,** 1591–1592
- `@Reference` **directive,** 14, 24, 1199
- **references**
  - building WCF consumer, 1370–1371
  - consuming XML Web services by adding, 1336–1337
  - tools, 1590–1591
- **Reflector,** 1602
- **Reformat Selection option,** 1452
- `regasm.exe` **(Assembly Registration Tool),** 1319, 1322–1323
- **RegExLib Web site,** 211
- **regions, localization.** ***See*** **localization**
- `@Register` **directive**
  - creating custom Web Part control, 849–850
  - defined, 14
  - overview of, 21–22
  - registering AJAX control on page, 936
- `RegisterClientScriptBlock` **method, server controls,** 84–86
- `RegisterClientScriptInclude` **method, server controls,** 88
- `RegisterStartupScript` **method, server controls,** 86–88
- `RegisterStartupScriptMethod()` **method, server controls,** 1222–1227
- **Registry Editor,** 1557–1559
- **Regular Expression Editor,** 210–211
- **RegularExpressionValidator server control,** 196, 209–211, 984–985
- **relational database structure, SQL Server,** 549
- **relative positioning, CSS,** 881–882
- **release configuration,** 1122–1123
- `ReleaseComObject` **class, memory,** 1309
- `ReleaseRequestState`**,** `Session` **object,** 1037

`RememberMeSet` **property, Login control,** 776–777
`RememberMeText` **property, Login control,** 776–777
Remote Debug Monitor (`msvsmon.exe`), 1126–1128
remote debugging, 1126–1128
remote servers, `web.config` **file,** 1447–1448
Remote Site, Copy Web Site GUI, 1535
remote sites, FrontPage Extensions, 5–6
`RemoveAccessRule()` **method,** 1164–1166
`RemovePreviousVersions` **property, Windows Installer,** 1554
`RemoveUserFromRole()` `method`**,** `Roles` **class,** 805–806
`Render()` **method, server controls,** 1222–1227
`RenderContents()` **method, overriding,** 1212
`RenderContents()` **method, server controlz,** 1209
rendering, ListView, 355–357
rendering HTML, server controls
  defined, 1210
  overview of, 1212–1214
  page event lifecycle, 1211–1212
`RepeatColumn` **property, CheckBoxList control,** 133–134
`RepeatColumn` **property, RadioButtonList control,** 137
`RepeatColumns` **property, DataList control,** 409
`RepeatDirection` **property**
  CheckBoxList control, 133–134
  DataList control, 409–410
  RadioButtonList control, 137
`RepeatLayout` **property, DataList control,** 406–407
Request Error Events Raised performance counter, 1470
Request Execution Time performance counter, 1470
request limit, for trace data, 1112
`Request` **object,** 1096–1097
request time-out, runtime, 1436–1437
Request Wait Time performance counter, 1470
`requestLengthDiskThreshold`**, runtime,** 1437
Requests Current performance counter, 1470
Requests Disconnected performance counter, 1470
Requests Failed performance counter, HTTP Status Codes, 1138
Requests Queued performance counter, 1470
Requests Rejected performance counter, 1470
RequiredFieldValidator server control, 196–202
  blank entries and, 201
  defined, 196
  overview of, 197–199
  using `InitialValue` property, 200–201
  validating drop-down lists, 201–202
  viewing results, 199–200
`requireSSL` **attribute, Forms authentication,** 1429
reserved folders, ASP.NET, 1571
ResizableControlExtender control, 972–974
`ResizableCssClass` **property, ResizableControlExtender control,** 974
Resource Editor, 1399, 1406–1407
resource files
  global, 1403–1406
  localization, 1397–1401
  .resx extension for, 712, 1397
  storing, 39
`resourceKey` **attribute,** `<siteMapNode>` **element,** 711
`ResponseMinimumDelaySeconds` **property, NoBot control,** 989
`Response.Write` **command,** 1390–1391
`RestartWWWService` **property, Windows Installer,** 1554
`.resx` **extensions (resource files),** 712, 1397
`Retrieve.aspx`**,** 1042–1046
`RetrieveTitle()` **method, consumer Web Parts,** 856
`RewritePath()` **method, URL rewriting,** 1286
role management service
  adding and retrieving application roles, 799–801
  adding users to roles, 802–803
  caching roles, 807–809
  checking users in roles, 806–807
  deleting roles, 801–802
  getting all roles of particular user, 805
  getting all users of particular role, 803–804
  membership management service vs., 799
  overview of, 757
  public methods of Roles API, 810
  removing users from roles, 805–806
  setting up Web site for role management, 796–799
  using LoginView server control, 793–795
  using Web Site Administration Tool, 808
role providers, 602–606
  `LimitedSqlRoleProvider`, 651–652
  overview of, 602
  `SqlRoleProvider`, 602–604
<RoleGroups>, LoginView control, 794–795
<rolemanager>, setting up Web site, 796–799
roles
  adding and retrieving application, 799–801
  adding users to, 802–803
  caching, 807–809
  checking users in, 806–807
  defining with Security Setup Wizard, 1503
  deleting, 801–802
  enabling security trimming, 718–720
  getting all users of particular, 803–804
  getting for particular user, 805
  IIS Manager and, 1520
  IIS-WebServerRole, 558
  managing, 715–717, 796–799, 1508–1509

  .NET Users, 1521–1522
  public methods of Roles API, 810
  removing users from, 805–806
  Wizard server control, 188–189
Roles API
  deleting end user's role cookie, 808–809
  public methods of, 810
  role management service with, 799–800
root element, XML documents, 499
root node, TreeView control, 676
`RootNode` **property,** `SiteMap` **class,** 706
`Rounded` **property, DropDownExtender control,** 955
RoundedCornersExtender control, 975–976
routing, events to SQL server, 1487–1490
`RowDataBound` **event, GridView control,** 319–321, 339–340
`RowDeleted` **event, GridView control,** 342–343
rows
  GridView control, 334–337
  Table server control, 140
`RowState` **property, GridView control,** 340
`RowUpdated` **event, DataGrid view,** 337–338
`RowUpdating` **event, GridView control,** 341
`RsaProtectedConfigurationProvider`**,** 621
RSS feeds
  viewing using ELMAH, 1598
  XmlDataSource control and, 308–309, 535–537
`RssCacheDependency` **class, custom cache dependencies,** 1083–1087
rules
  adding/removing ACL, 1163–1166
  CSS. *See* HTML and CSS design, CSS rules
  health monitoring, 1483–1485
  managing for access in Security tab, 1510
  routing events to SQL server, 1488–1489
  Windows Installer, 1540
  XML, 1326
Run As Server Control, Visual Studio, 76
`runat='server''` **attribute declaration**
  creating skin using, 270–271
  HTML server controls requiring, 76–79
  Web server controls requiring, 108
`RunPostBuildEvent` **property, Windows Installer,** 1554
runtime
  applying configuration files, 1414–1415
  configuration settings, 1436–1438
  loading user controls dynamically at, 1198–1203
  server controls rendering HTML at, 1210–1214
Runtime Callable Wrapper. ***See*** **RCW (Runtime Callable Wrapper)**

S

salted hash, 1066
`Save` **method,** `XmlDataSource` **control,** 309
ScaleOut Software, 1057
schema, XML
  adding in SQL Server 2005, 552–554
  associating XML typed column with, 554
  editing, 502–506
  using with `XmlTextReader`, 509–511
  using XML Schema Definition (XSD), 501–502
  validating against with `XDocument`, 511–513
`schemaLocation` **attribute, editing XML and XML schema,** 503–506
`ScriptControl` **class,** 1203
ScriptManager server control, AJAX, 259–262, 912–915
ScriptManagerProxy server control, AJAX, 261–262, 912, 914–916
scripts, 747, 1222–1227
scrollbars, Panel server control, 154
`SearchPath` **property, Windows Installer,** 1554
security
  authentication. *See* authentication
  authorization. *See* authorization
  Basic authentication, 1004–1005
  client-side vs. server-side validation and, 194–195
  Forms-based authentication. *See* Forms-based authentication
  identity and impersonation, 1023–1025
  I/O, 1139
  membership management. *See* membership management service
  .NET Trust Levels, 1520–1521
  overview of, 995–996
  Passport authentication. *See* Passport authentication
  personalization. *See* personalization
  programmatic authorization. *See* authorization, programmatic
  protecting configuration settings, 1448–1452
  role management. *See* role management service
  through IIS, 1025–1032
  Windows-based authentication, 998–1006
Security event log, 1465
Security Setup Wizard, 1502–1507
Security tab, Web Site Administration Tool
  creating users, 1507–1508
  enabling role management in, 715–716
  managing access rules, 1510
  managing roles, 1508–1509
  managing users, 1508
  overview of, 1501–1502
  Security Setup Wizard, 1502–1507
security trimming, 714–720
  enabling, 718–720
  overview of, 714–715
  setting up administrators' section, 716–717

security trimming *(continued)*
  setting up role management for administrators, 715–716
`securityTrimmingEnabled` **attribute, XmlSiteMapProvider,** 720
Select a Master Page dialog, 235–239
SELECT statement
  LINQ query syntax, 466
  SqlDataSource control connections, 292
selecting data, from SQL database, 378–380
`SelectionMode` **attribute, Calendar control,** 144
`SelectionMode` **attribute, ListBox control,** 126
Selectors, CSS
  combinations, 872
  grouping, 872
  merging styles, 872–875
  overview of, 867–869
  working with CSS in Visual Studio, 886–887
`SelectParameters`**, DetailsView server control,** 349
`SelectParameters` **property, SqlDataSource control,** 294–296
semantics, XML syntax vs., 500–501
SeparatorTemplate, DataList control, 407–409
serial ports, 1181–1182
`Serializable` **attribute**
  Out-of-Process Session State, 1047–1051
  ViewState, 1064
serialization, XML, 516–517
server configuration files, 1411–1413
server controls. *See also* validation server controls
  AJAX. *See* AJAX (Asynchronous JavaScript and XML), server-side controls
  AJAX Control Toolkit. *See* AJAX Control Toolkit server controls
  attributes of, 1209–1211
  building pages with, 65–67
  client-side callback, working with. *See* client-side callback
  client-side features, adding to, 1222–1231
  composite controls, 1244–1247
  control designers, 1258–1271
  defined, 1193
  detecting and reacting to downlevel browsers, 1231–1234
  events, 67–70
  HTML, 76–83
  HTML, styling, 1217–1220
  manipulating with JavaScript, 83–88
  overview of, 63–64, 1203
  postback data, 1242–1244
  postback events, 1238–1242
  rendering, 1210–1214
  skins, 269–271, 1220–1221
  styles applied to, 70–75
  tag attributes, 1214–1217
  templated, 1247–1254
  themes, 266–267, 1220–1221
  types of, 64–65
  UI type editors, 1271–1273
  using ControlState, 1236–1238
  using type converters, 1254–1258
  using ViewState, 1234–1236
  WebControl project setup, 1204–1209
Server Explorer, event logs. *See* event logs
Server Extensions, ASP.NET AJAX, 901–902
servers
  routing events to SQL, 1487–1490
  state management options, 1035
server-side technologies
  AJAX, 900–902
  authentication, 1427
  culture declarations, 1386–1387
  validation, 213–216
service contract, WCF service
  creating interface for, 1365–1366
  defined, 1364
  implementing interface for, 1366–1367
  setting namespace, 1379
`Service.asmx` **file,** 1327–1329
`ServiceMethod` **attribute, DynamicPopulateExtender control,** 958
service-oriented architecture (SOA), 1360–1362
services
  WCF. *See* WCF (Windows Communication Foundation)
  XML Web. *See* XML Web services
`Session` **object,** 1036–1059
  in classic ASP, 1036
  configuring session state management, 1038
  cookieless session state and, 1057–1058
  event model and, 1036–1038
  extending session state with other providers, 1056–1057
  in-process session state and, 1038–1043
  maintaining, 1058–1059
  making transparent, 1043–1045
  optimizing performance, 1045–1046
  Out-of-Process session state, 1046–1051
  SQL-backed session state, 1051–1056
session state
  configuration files, 1410
  configuration settings, 1410, 1417–1421
  configuring management, 1038
  cookieless, 1057–1058
  extending with other providers, 1056–1057
  in IIS Manager, 577–579, 1524–1526
  In-Process, 1038–1043
  Out-of-Process, 1046–1051
  provider model in ASP.NET 3.5 for, 589–591
  providers, 588–589, 609–612
  Session State Settings for, 1525–1526
  SQL-backed, 1051–1056
  storing, 587–588
`SessionStateModule`**,** 1056
`SessionStateStoreProviderBase`**,** 1056

Set-Cookie HTTP Header, state, 1034
`SetCurrentDirectory( )` **method,** `Directory` **class,** 1147–1148
SetFocusOnError property, validation groups, 226
`SetItemExpireCallback( )` **method,** `SessionStateModule`, 1056
Setup Project, Windows Installer service, 1541
Setup Wizard, Windows Installer service, 1541
SGML (Standard Generalized Markup Language), 499, 1326
`Shared` **classes, ADO.NET,** 383
`Shell` **command,** 1106
shortcuts, desktop, 1557
`ShowCheckBoxes` **property, TreeView control,** 679–683
`ShowLines` **property, TreeView control,** 685–687
`ShowStartingNode` **property, SiteMapDataSource control,** 703–704
`ShowToolTips` **property, SiteMapPath server control,** 669–670
Silverlight, 1607–1626
  accessing from JavaScript events, 1625–1626
  basic ASP.NET application, 1609–1610
  converting vector content to XAML, 1611–1613
  extending application with, 1607–1608
  integrating with existing ASP.NET site, 1620–1623
  overview of, 1607
  receiving events in JavaScript, 1623–1625
  vector-based content in, 1610–1611
  viewing and editing XAML, 1613–1620
`SimpleMailWebEventProvider`, 613, 615–617, 1493–1494
`SingleTagSectionHandler`, 1453, 1457–1458
site maps
  defined, 661
  localization of, 710–714
  nesting, 720–722
  URL mapping, 709–710
  using SiteMapDataSource control, 703–706
  using SiteMapPath control. *See* SiteMapPath server control
  using TreeView control. *See* TreeView server control
  XML-based, 662–663
site navigation, 661–722
  nesting `.sitemap` files, 720–722
  security trimming and, 714–720
  using Menu server control. *See* Menu server control
  using SiteMap API, 706–709
  using sitemap localization, 710–714
  using SiteMapDataSource control, 703–706
  using SiteMapPath control, 664–670
  using TreeView server control. *See* TreeView server control
  using URL mapping, 709–710
  using XML-based site maps, 662–663
SiteMap API, 706–709
`SiteMap` **class,** 661, 662, 706
.sitemap file. ***See*** **site maps**
SiteMap provider, 608–609
SiteMapDataSource control
  applying to Menu control, 693–694
  applying to TreeView control, 673
  data source controls, 314
  interacting with site maps using, 662
  overview of, 703
  `ShowStartingNode` property, 703–704
  `StartFromCurrentNode` property, 704–705
  `StartingNodeOffset` property, 705
  `StartingNodeUrl` property, 706
  testing site map localization results, 712–714
`<siteMapNode>` **element, XML-based site maps,** 662–663
SiteMapPath server control, 664–670
  child elements of, 670
  creating own style for, 708–709
  overview of, 664–666
  `ParentLevelsDisplayed` property, 669
  `PathDirection` property, 668
  `PathSeparator` property, 666–668
  `ShowToolTips` property, 669–670
Sites node, IIS Manager, 575–576
size limitations, files, 167–170
`SkinID` **attribute,** 279–280, 281
skins
  creating for server controls, 269–271
  creating multiple options for, 278–280
  creating server controls with, 1220–1221
  in custom controls, 281–285
  incorporating images into themes using, 276–278
`Skip` **method, LINQ,** 475–476
SliderExtender control, 976–977
SlideShowExtender control, 977–979
`slidingExpiration` **attribute, Forms authentication,** 1429
smart controls, 64
Smart Device Cab Project, Windows Installer, 1541
smart tags
  server control, 1269–1271
  syntax notification at design-time, 1105–1106
`smartNavigation` **element,** 1434
SMTP
  configuring in Application tab, 1511
  IIS Manager, 1526–1527
  set-up for e-mailing Web events, 1494
SOA (service-oriented architecture), 1360–1362
SOAP (Simple Object Access Protocol),
  caching responses, 1350–1351

SOAP (Simple Object Access Protocol) *(continued)*
  communicating with XML Web service using, 1345–1346
  defining XML structure through, 1326
  exposing custom datasets as, 1330–1333
  requests, 1326–1327
  responses, 1327
  XML Web service interface displaying messages, 1334–1335
SOAP headers, 1350–1357
  building Web services with, 1351–1353
  consuming Web services with, 1353–1355
  overview of, 1350–1351
  requesting Web services with SOAP 1.2, 1355–1357
SortByCategory, page-level tracing, 1108
SortByTime, page-level tracing, 1108
sorting data
  adding to GridView control, 323–325
  strings in different cultures, 1394–1397
  in traditional query methods, 462–464
sounds, in error notification, 221–222
Source view
  coding server controls in, 67
  GridView control columns in, 331
  HTML server controls in, 76–79
`SourceSwitch`**,** 1120–1121
`Span` **tag, server controls,** 1209
specific culture definitions
  currency translation, 1394
  defined, 1382
  vs. neutral cultures, 1403
SQL Server
  debugging, 1134
  setting up providers for versions of, 591–598
  writing Web events to, 618–619
SQL Server, 2000
  cache dependency, 1089–1090
  cache invalidation, 1097
  caching in, 1089–1091
  creating users with SqlMembershipProvider, 629
  locating Northwind.mdf file in, 378
  membership provider for, 598, 600
  personalization provider, 746–748
  personalization provider for, 606
  retrieving XML from, 546–547
  role provider for, 602
  setting up provider to work with, 591–598
  SQL stored proc debugging, 1134
  support for XML on, 544
  Web Parts provider for, 623
SQL Server, 2005
  cache invalidation, 1091–1092, 1097
  connecting role management system to, 604
  creating users with SqlMembershipProvider, 629
  debugging, 1134
  membership provider for, 598, 600
  personalization provider for, 606–608, 746–748
  `pollTime` attribute and, 1093
  role provider for, 602
  routing events to SQL server, 1488
  setting up provider to work with, 591–598
  SQL stored proc debugging, 1134
  SQL to LINQ generating SQL optimized for, 487
  Web Parts provider for, 623
  writing Web events to, 618–619
SQL Server 2005, and `XML` **data type,** 549–556
  adding column of untyped XML, 551–552
  adding XML schema, 552–554
  associating XML typed column with schema, 554
  generating custom XML, 550–551
  inserting XML data into XML column, 554–556
  overview of, 549–550
SQL Server, 2008
  creating users with SqlMembershipProvider, 629
  membership provider for, 600
  personalization provider for, 606, 746–748
  role provider for, 602
  setting up provider to work with, 591–598
  Web Parts provider for, 623
SQL Server, 7.0
  cache invalidation, 1097
  personalization provider, 746–748
  setting up provider to work with, 591–598
SQL Server Cache Dependency, 1087–1092
  cache invalidation, 1091–1092
  disabling databases for cache invalidation, 1091
  disabling table for cache invalidation, 1090–1091
  enabled tables, 1090
  enabling databases for cache invalidation, 1088
  enabling table for cache invalidation, 1088–1089
  overview of, 1087–1088
  and SQL Server 2000, 1089–1090
SQL Server cache invalidation. *See also* SQL Server Cache Dependency
  overview of, 1092–1093
  testing, 1094–1101
SQL Server Express Edition (`.mdf` **file)**
  adding users to membership service, 761–775
  personalization provider, 606–608, 745–746
  providers working with, 591
  role providers, 602–606
  `SqlMembershipProvider` in, 600
  Web site setup for membership, 758–761
SQL Server scripts, 747
SQL* Plus, 401
`SqlCacheDependency` **class**
  attaching SQL cache dependencies to `Request` object, 1096–1098

  configuring ASP.NET application, 1093
  defined, 1087
`SqlCommand` **class,** 386–389, 433–434
`SqlConnection` **class**
  asynchronous connections, 454
  overview of, 384–385
  using `DataReader` object, 387–389
`sqlConnectionString` **attribute,** 1418
`SqlDataAdapter` **class,** 389–392
SqlDataSource control
  adding `UpdateCommand` to, 335
  configuring data connection, 289–293
  `ConflictDetection` property, 296–297
  `DataSourceMode` property, 293
  events, 297–299
  filtering data using `SelectParameters`, 294–296
  overview of, 289–301
`SqlDependency` **object,** 1094–1100
SqlMembershipProvider, 775
`SqlMembershipProvider` **class**
  building providers, 635
  overview of, 598–600
  simpler password structures, 629–632
  stronger password structures, 632–633
  Web site set up for membership, 758–761
`SqlParameter` **class,** 392–395
`SqlPersonalizationProvider`**,** 623–625
`SqlProfileProvider`**,** 606–608, 747–748
`SqlRoleProvider` **class,** 602–604
  `AddUsersToRole( )` method, 655–656
  `CreateRole( )` method, 653–655
  `DeleteRole( )` method, 655
  role management service with, 796–799
  working with `LimitedSqlRoleProvider`, 651–652
`SQLServer`
  provider, 588
  session state configuration, 1419–1421
`SqlSessionStateStore`
  configuring `sessionState` management, 1038
  defined, 610
  working with, 612
`SqlWebEventProvider`
  buffering Web events, 1492
  defined, 613
  overview of, 618–619
squiggles (syntax notifications), 1103–1106
Standard Generalized Markup Language (SGML), 499, 1326
`Start` **events,** `Session` **object,** 1037
`StartFromCurrentNode` **property, SiteMapDataSource control,** 704–705
`StartingNodeOffset` **property, SiteMapDataSource control,** 705
`StartingNodeUrl` **property, SiteMapDataSource control,** 706
state management, 1033–1069
  `Application` object, 1059
  ControlState, 1067
  cookies, 1060
  deciding on method for, 1034–1036
  hidden fields, 1063–1065
  history of, 1033
  postbacks and cross-page postbacks, 1061–1063
  QueryStrings, 1060
  using `HttpContext.Current.Items` for short-term storage, 1067–1068
  ViewState, 1063–1066
state management, `Session` **object in,** 1036–1059
  choosing correct way to maintain, 1058–1059
  configuring `sessionState` management, 1038
  cookieless session state, 1057–1058
  event model and, 1036–1038
  extending session state with other providers, 1056–1057
  in-process session state, 1038–1043
  making transparent, 1043–1045
  optimizing performance, 1045–1046
  Out-of-Process Session State, 1046–1051
  SQL-backed session state, 1051–1056
`State` **property,** `SqlConnection` **class,** 454
State Server Sessions Abandoned performance counter, 1470
State Server Sessions Active performance counter, 1470
State Server Sessions Timed Out performance counter, 1470
State Server Sessions Total performance counter, 1470
State Service, 1047, 1050–1051
`stateConnectionString` **attribute,** `<sessionState>`**,** 1418
`stateNetworkTimeout` **attribute,** `<sessionState>`**,** 1418
`StateServer`**,** 588, 1418–1419
static links, Menu control styles for, 695–696
`StaticBottomSeparatorImageUrl` **property, Menu control,** 699–700
<StaticHoverStyle>, Menu control, 696
<StaticMenuItemStyle>, Menu control, 696
`StaticPopOutImageUrl` **property, Menu control,** 699
`StaticTopSeparatorImageUrl` **property, Menu control,** 699–700
`StepType` **attribute, Wizard server control,** 180–181
stored procedures, 488–490, 491–493

storing
  application-specific settings, 1440–1441
  connection strings, 1416–1417
  session state, 1417–1421
StrangeLoop Network, AppScaler, 1057
`Stream` **class**
  classes derived from, 1167
  compressing, 1176–1181
  compressing streams, 1176–1181
  encodings, 1174–1175
  I/O shortcuts, 1175–1176
  overview of, 1167–1171
  `Reader` and `Writer` classes, 1171–1174
  reading and writing I/O data with, 1166
`Stream` **object,** 172–173
`StreamReader`**,** 1172–1173, 1174–1175
`StreamWriter`**,** 1175
`strict` **attribute, compilation configuration,** 1423
strings, sorting, 1394–1397
Style Builder, 73–74
<style> tag, HTML, 865–866
styles
  adding to `PathSeparator` property, 667
  BulletedList server control, 158
  Calendar server control, 147–148
  changing with HTML elements, 72
  DataList control template, 408–409
  displaying in Manage Style Tool window, 888
  GridView control, 343–344
  HTML for server controls, 1217–1220
  limitations of HTML for, 862
  Login control, 779
  Menu control, 694–700
  TreeView control, 674–675
  using Apply Styles tool window, 888
  using CSS for. *See* CSS (Cascading Style Sheets)
  using themes. *See* themes
  watermark, 981
`StyleSheetTheme` **attribute,** `Page` **directive,** 268
`Subject` **property, Windows Installer,** 1554
submaster pages, 254–256
`Subscribers` **group,** `IsInRole` **method,** 1019–1020
Substitution Control, Post-Cache, 1075–1077
`SupportPhone` **property, Windows Installer,** 1554
`SupportsEvaluate` **property, expression builders,** 373–374
`SupportUrl` **property, Windows Installer,** 1554
`Switch` **class**
  `aspnet_regIIS.exe` utility, 1439
  `BooleanSwitch`, 1119–1120
  for diagnostic switches, 1119
  `SourceSwitch`, 1120–1121
  `TraceSwitch`, 1120
Synclock statement, 650
syntax
  notifications (squiggles), 1103–1106
  VB in Visual Studio 2008 vs. C#, 508
  XML semantics vs., 500–501
System event log, writing to, 1465
`System.AccessControl` **namespace,** 1160–1161
`System.ComponentModel.TypeConverter` **class,** 1256
`System.Configuration`**,** 620–623, 1441–1448, 1453–1460
`System.Core.dll` **assembly, LINQ,** 464–465
`System.Data.OracleClient`**,** 400–403
`System.Diagnostics`**,** 1462–1464, 1470–1476
`System.Diagnostics.Trace`**,** 1108, 1113–1116
`System.Exception`**,** 1136–1137
`System.IO` **namespace,** 1140–1148
`System.IO.Compression`**,** 1176–1177
`System.IO.Path`**,** 1154–1158
`System.IO.Ports`**,** 1181–1182
`System.Net` **namespace**
  defined, 1182
  `FileWebRequest` and `FileWebResponse` classes, 1188–1189
  `FtpWebRequest` and `FtpWebResponse` classes, 1186–1188
  `HttpWebRequest` and `HttpWebResponse` classes, 1183–1186
  sending mail, 1189–1190
`System.Net.Mail` **namespace,** 1189–1190
`SystemWeb.config`**,** 1435
`System.Web.Configuration`**,** 1441–1448
`System.Web.HttpBrowserCapabilities`**,** 1233–1234
`System.Web.Mail`**,** 1189
`System.Web.Management`
  event types, 1481–1482
  Web event providers, 612–613
`System.Web.Management.EventLogWebEventProvider`**,** 613–615, 1478
`System.Web.Management.IISTraceWebEventProvider-`**,** 613, 1478
`System.Web.Management.SimpleMailWebEventProvider`**,** 613, 615–617, 1478
`System.Web.Management.SqlWebEventProvider`**,** 613, 618–619, 1478
`System.Web.Management.TemplatedMailWebEventProvider`**,** 613, 617–618, 1478
`System.Web.Management.TraceWebEventProvider`**,** 613, 619, 1478
`System.Web.Management.WmiWebEventProvider`**,** 613, 619–620, 1478
`System.Web.Script.Services.ScriptService`**,** 1330
`System.Web.Security.ActiveDirectoryMembershipProvider`**,** 600–602
`System.Web.Security.AuthorizationStoreRoleProvider`**,** 605–606

`System.Web.Security.SqlMembershipProvider`, 598–600
`System.Web.Security.SqlProfileProvider`, 606–608
`System.Web.Security.SqlRoleProvider`, 602–604
`System.Web.Security.WindowsTokenRoleProvider`, 604–606
`System.Web.Services.Protocols`, 1330
`System.Web.SessionState`, 609–612
`System.Web.UI.ICallbackEventHandler` **interface,** 1227–1231
`System.Web.UI.ScriptControl`, 1203
`System.Web.UI.WebControls`, 1203
`System.Web.UI.WebControls.WebParts.SqlPersonalizationProvider`, 623–625

T

TabContainer control, AJAX, 993–994
Table menu, Visual Studio Design view, 884–885
Table server control, 139–141
TableAdapter Configuration wizard, 424–426
`TableAdapter` **object,** 423–424, 432
tables, 1088–1091, 1096
TabPanel controls, 993–994
tags
- rendering HTML, 1212–1214
- rendering HTML attributes, 1214–1217
- user control, 1194

`Take` **method, LINQ,** 475–476
`TargetControlID` **property**
- AlwaysVisibleControlExtender control, 938
- AnimationExtender control, 940
- CollapsiblePanelExtender control, 947
- DragPanelExtender control, 951
- DropDownExtender control, 953
- DropShadowExtender control, 954
- FilteredTextBoxExtender control, 960
- ListSearchExtender control, 963
- MaskedEditExtender control, 964
- ResizableControlExtender control, 973
- RoundedCornersExtender control, 975

`TargetPlatform` **property, Windows Installer,** 1554
Task List views, 1106–1107
Temp DB database, 1420
`tempDirectory` **attribute, compilation configuration,** 1423
templated controls, 1247–1254
`TemplatedMailWebEventProvider`, 613, 617–618, 1495–1497
TemplateField column, GridView control, 331–334
TemplateField EditItem template, GridView control, 339–341
TemplatePagerField, ListView paging, 359–360
templates. ***See also*** **master pages**
- AJAX Control Toolkit, 931–932
- ASP.NET Server Control class, 1204–1206
- controls, 322
- DataList server control. *See* DataList server control
- GridView control, 322
- ListView. *See* ListView server control, templates

testing
- health monitoring, 1486–1487
- migration from ASP.NET 1x to 2.0, 1569
- site map localization, 712–714
- SQL Server cache invalidation, 1094–1101

text
- aligning check box, 131
- compiling, 44
- Hyperlink control, 120–121
- Label control, 108–110
- Literal control, 110–111
- TextBox control, 111–115
- UpdateProgress control, 922–925

`Text` **property**
- Regular Expression validator control, 210
- RequiredFieldValidator control, 198–199
- `TreeNode` object, 690–693
- validation control with, 217–218
- ViewState, 1235–1236

`TextAlign` **property, CheckBox control,** 131
TextBox server control, 111–115
- with FilteredTextBoxExtender control, 959–961
- with NumericUpDownExtender control, 968–969
- overview of, 111–112
- with PasswordStrength control, 990–991
- with SliderExtender control, 976–977
- with TextBoxWatermarkExtender control, 979–981
- using `AutoCompleteType`, 114–115
- using `AutoPostBack`, 113
- using `Focus()` method, 112

`TextBoxStringProvider()` **method, provider Web Parts,** 854
TextBoxWatermarkExtender control, 979–981
`Themeable` **attribute,** 282–285
themes, 263–286
- assigning skin programmatically, 281
- assigning to entire application, 265
- assigning to page programmatically, 280–281
- assigning to single ASP.NET page, 263–265
- creating proper folder structure, 268–269
- creating server controls with, 1220–1221
- creating skin, 269–271
- custom controls, skins and, 282–285
- defining multiple skin options, 278–280
- folder for storing, 38

themes *(continued)*
  master pages using, 267–268
  removing from server controls, 266–267
  removing from web pages, 267
  `StyleSheetTheme` attribute, 268
  using CSS files in, 272–275
  using images in, 275–278
thick-client applications, 895–896
thin-client applications, 895–896
third-party vendors
  DbgView, 1113
  extending LINQ, 493–494
  GenerateMachineKey tool, 1066
  HTTP compression modules, 1178
  HTTP Headers and controlling caching, 1078
  session state providers, 1056–1057
  validation server controls, 196
  ViewStateDecoder tool, 1066
threads, ASP.NET, 1383–1386, 1437
time, culture differences, 1387–1391
timeouts
  Forms authentication, 1429
  runtime settings configuration, 1436–1437
  session state configuration, 1418
  Session State Settings, 1526
Timer server control, AJAX, 912, 916–917
titles
  content page, 242–243
  custom content page, 242–243
  Windows Installer, 1554
`.tlb` **file extension (type library file),** 1315–1316
`tlbexp.exe`**,** 1315–1319
ToggleButtonExtender control, 982–983
tools, 1583–1605
  configuration file editing, 1452–1453
  debugging, 1583–1589
  extending ASP.NET, 1597–1599
  general purpose developer, 1600–1604
  IIS Manager. *See* IIS (Internet Information Service) Manager
  references, 1590–1591
  resource editor, 1406–1407
  tidying up code, 1591–1594
  Visual Studio add-ins, 1594–1596
  Web Site Administration Tool. *See* Web Site Administration Tool
  Windows Installer service. *See* Windows Installer service
ToolTips
  SiteMapPath server control, 669–670
  for syntax errors at design-time, 1104
  using HoverMenuExtender, 961–962
`ToShortDateString( )` **method, Calendar control,** 144
trace forwarding, 1114
`Trace` **property,** `Page` **class,** 1108, 1109
`<trace>` **setting, locking-down configuration,** 1433
`trace.axd`**,** 1109–1112
`TraceContext` **class,** 1108
`Trace.IsEnabled` **property, page-level tracing,** 1108
TraceListeners
  configuring, 1114–1116
  defined, 1107–1108
  DelimitedListTraceListener, 1118–1119
  EventLogTraceListener, 1116–1118
  IISTraceListener, 1119
  listening in on debugging with, 1113
  new WebPageTraceListener, 1116
  XmlWriterTraceListener, 1118–1119
TraceMode attribute, page-level tracing, 1108
Tracepoints, 1131
`TraceSwitch`**,** 1120
`TraceWebEventProvider`**,** 613, 619
`Trace.Write` **function,** 1109, 1113–1114
tracing, 1107–1122
  application, 1108, 1476
  ASP.NET's `Page.Trace`, 1108
  from components, 1113–1114
  configuring TraceListeners, 1115–1116
  DelimitedListTraceListener, 1118–1119
  diagnostic switches, 1119–1121
  EventLogTraceListener, 1116–1118
  managing in Application tab, 1511–1512
  new ASP.NET WebPageTraceListener, 1116
  overview of, 1107–1108
  page-level, 1108
  storing data in `Session` object, 1041–1042
  `System.Diagnostics.Trace`, 1108
  trace forwarding, 1114
  viewing trace data, 1109–1112
  Web events, 1121–1122
  XmlWriterTraceListener, 1118–1119
`Transform` **method,** `XslCompiledTransform` **class,** 538–541
translation
  dates, 1389–1391
  number and currency, 1391–1394
  sorting strings, 1394–1397
transport protocols, Web services, 1341–1346
`TreeNode` **objects, TreeView control,** 690–693
TreeView class, TreeView control, 687–693
TreeView Line Generator dialog, 686
TreeView server control, 670–693
  binding to XML file, 676–679
  built-in styles of, 674–675
  as databound control, 366
  incorporating images into themes, 275
  overview of, 670–674
  parts of, 676
  programmatically working with, 687–693
  selecting multiple options, 679–683

  - specifying custom icons, 683–684
  - specifying lines used to connect nodes, 685–687
  - styling with CSS-Friendly Control Adapters, 893
  - testing site map localization results, 712–714
- `TreeView1_DataBound( )` **method,** 689–690
- `TreeViewLineImages` **property, TreeView control,** 687
- **triggers,** 918–922, 926
- **Triple DES encryption,** 623
- **trust levels**
  - IIS Manager, 1520–1521
  - security and, 1025
  - server configuration files, 1412–1413
- `Try/Catch`**, in exception handling,** 1134–1135
- **two-way data binding,** 368
- **type converters,** 1254–1258
- **type libraries,** 1314–1316
- **Type Library Exporter,** 1315–1316, 1319
- **Type Library Importer,** 1303–1304, 1307
- `Type` **property, CompareValidator server control,** 205
- `Type` **property, RangeValidator server control,** 206–207
- **Type Selectors, CSS,** 867
- **typed datasets,** 33–38, 400
- **types**
  - personalization properties, 731–733
  - ViewState and, 1236

U

- **UI type editors,** 1271–1273
- `uiculture` **attribute,** `Web.config` **file,** 711
- `uiculture` **attribute,** `Web.config.comments` **file,** 1386–1387
- `unattend.xml` **file,** 568
- **unhandled exceptions,** 1135, 1136
- **Uniform Resource Locator.** ***See*** **URL (Uniform Resource Locator)**
- **Uniform/Universal Resource Identifier (URI),** 499, 738
- **uninstalling applications, Windows Installer,** 1546–1547
- **unique identifiers (GUIDs),** 243–245, 738–739
- `UniqueID` **property, server controls,** 1217
- **United Kingdom, culture codes,** 1382
- **United States, culture codes,** 1382
- **Universal Selectors, CSS,** 867
- `Unlock( )` **method,** `Application`**,** 1059
- **unmanaged code.** ***See*****.NET from unmanaged code**
- `UnsealedCacheDependency` **class,** 1082–1083
- `UpdateCommand`
  - DetailsView control, 350–351
  - SqlDataSource control, 335
- **updateGrid ( ) function, DynamicPopulateExtender control,** 958
- `UpdateMode` **property, UpdatePanel,** 927
- **UpdatePanel server control, AJAX,** 917–922
- **UpdatePanelAnimationExtender control,** 983–984
- **UpdatePanelTrigger Collection Editor,** 921
- `UpdateParameters`**, SqlDataSource control,** 335
- **UpdateProgress server control, AJAX,** 912, 922–925
- **updates**
  - browser configuration, 1425
  - in DetailsView control, 349–350
  - handling errors in GridView control during, 337–339
  - IIS 7 dependencies and, 565
  - LINQ to SQL, 493
  - in SQL Server, 381–382
  - using AJAX on Web page, 897
  - using `pkmgr.exe` to install specific, 568
- `UpgradeCode` **property, Windows Installer,** 1554
- **upgrades**
  - to ASP.NET 3.5 using VS 2008, 1570
  - to ASP.NET applications, 1568–1570
  - to IIS 7, 569
  - using ASP.NET reserved folders, 1571
  - using stepped approach in, 1570
- **upload buffering,** 1437
- **uploading files.** ***See*** **FileUpload server control**
- **URI (Uniform/Universal Resource Identifier),** 499, 738
- **URL (Uniform Resource Locator)**
  - installation, 1551–1552
  - mapping, 709–710
  - state management and, 1034–1035
  - XML namespaces as, 499
- **URL Authorization,** 1431–1432
- **URL rewriting,** 1283–1286
- `URLAuthorizationModule`, 1431–1432
- `UseCookies` **value, storing identifiers,** 738
- `UseDeviceProfile` **value, storing identifiers,** 738
- `useFullyQualifiedRedirectUrl` **attribute, runtime,** 1436
- `useMachineContainer` **attribute,** `RsaProtectedConfigurationProvider`**,** 623
- `useMachineProtection` **attribute,** `DpapiProtectedConfigurationProvider`**,** 622
- `useOAEP` **attribute,** `RsaProtectedConfigurationProvider`**,** 623
- **user controls,** 1193–1203
  - creating, 1194–1196
  - defined, 1193
  - interacting with, 1196–1198
  - loading dynamically, 1198–1203
  - problems with in ASP.NET 1.0, 231
- **User Interface Editor,** 1561–1562
- `User` **property,** `Page` **object,** 1017–1018
- **User Tasks view, Task List,** 1106–1107

`userControlBaseType` **element, `<pages>` configuration,** 1434
UserControls (ASCX), 1074–1075
`User.Identity` **property,** 1017–1019
`User.IsInRole( )` **method,** 1018–1020
usernames
  adding to membership service, 761–763
  authenticating against values in database, 1012–1014
  authenticating against values in `web.config` file, 1010–1012
  Basic authentication limitation, 1004
  locking out users with bad passwords, 780–784
  with LoginName server control, 784–786
  protecting configuration settings, 1448
  using Login control with Forms authentication, 1014–1015
users
  adding with Security Setup Wizard, 1503, 1506
  allowing to change Web page mode, 820–833
  authentication. *See* authentication; membership management service
  authorization. *See* authorization; role management service
  creating and placing in Admin role, 715–716
  membership providers validating, 647–648
  .NET Users, IIS, 1521–1522
  personalization properties for. *See* personalization properties
  rating control manipulated by, 991–992
  `ReadUserFile( )` method, 648–651
  using `XmlMembershipProvider` for login, 651–652
  Windows authentication. *See* Windows-based authentication
  working in Security tab, 1507–1508
`UseURI` **value, storing identifiers with,** 738
`Using` **statement**
  closing streams with, 1168
  `XmlReader` implementing, 525, 549
  `XmlWriter` implementing, 525

V

`ValidateNumber` **function, CustomValidator control,** 215
`validateRequest` **element, `<pages>` configuration,** 1435
`ValidateUser( )` **method**
  `Membership` class, 779–780
  `MembershipProvider` class, 647–648
validation
  CustomValidator control, 211–216
  Forms authentication and, 1429
  MaskedEditValidator control, 966
  with membership providers, 647–648
  against schema with `XDocument`, 511–513
  ValidatorCalloutExtender control, 984–985
  of XML, 501–502, 506–509
Validation and More (VAM) server control, 196
validation server controls, 193–228
  client-side vs. server-side, 194–195
  CompareValidator, 202–206
  CustomValidator, 211–216
  defined, 193
  as extender controls, 937
  overview of, 195–196
  RangeValidator, 206–209
  RegularExpressionValidator, 209–211
  RequiredFieldValidator, 196–202
  turning off client-side validation, 220–221
  understanding validation, 193–194
  using images and sounds for error notifications, 221–222
  validation causes, 196
  ValidationSummary, 216–220
  working with validation groups, 222–227
`ValidationExpression` **property, Regular Expression validator control,** 210–211
`ValidationGroup` **property, validation groups,** 223–225
`<validationkey>`, `web.config` **file,** 1066
ValidationSummary server control, 196, 216–220
ValidatorCalloutExtender control, 984–985
`Value` **property,** `TreeNode` **object,** 690
`ValueChanged` **event, HiddenField control,** 163–164
value-required rule, RequiredFieldValidator control, 197
`ValueToCompare` **property, CompareValidator control,** 205–206
VAM (Validation and More) server control, 196
`VaryByControl` **attribute, output caching,** 1073–1074
`VaryByCustom` **attribute, output caching,** 1073–1074
`VaryByParam` **attribute, output caching,** 1072–1073, 1094
vector graphics, 1610–1613
verbs
  URL Authorization configuration, 1431–1432
  Web Part, 836–838
`Version` **property, Windows Installer,** 1554
View server control, 174–178
views
  querying LING to SQL data using, 487–488
  of trace data, 1109–1112

ViewState
  client-side state management, 1036
  state management, 1063–1066
  tips for, 1066
  using ControlState in server controls, 1236–1238
  using in server controls, 1234–1236
ViewStateDecoder tool, 1066
virtual directories, 1415
`VirtualDirectory` **property, File System Editor,** 1556
`visibility` **property, CSS box model,** 879–880
Visibone Web site, 1590
Visual Studio
  ADO.NET tasks with, 419–432
  classes, working with, 54–61
  configuration file editing tools, 1452–1453
  Copy Web Site deployment option, 1534–1537
  global resource files, 1404–1406
  HTML and CSS in. *See* HTML and CSS design, in Visual Studio
  HTML server controls, 76–79
  precompilation for deployment, 1538–1539
  Resource Editor, 1406–1407
  Style Builder, 73–74
Visual Studio, 2008
  add-in tools, 1594–1596
  AJAX, working with. *See* AJAX (Asynchronous JavaScript and XML)
  AJAX Control Toolkit and. *See* AJAX Control Toolkit
  ASP.NET 1.x applications, converting, 1574–1579
  ASP.NET 2.0 to 3.5, migrating from, 1580–1582
  ASP.NET 3.5, upgrading application to, 1570
  ASP.NET projects, creating new, 1305–1307
  configuration files, editing, 1452
  CSS, working with, 884–889
  CSS Attribute Selectors not supported in, 868
  deploying applications, 1532–1537
  design surface, 882, 1252
  event log, working with, 1459–1468
  first-child pseudo class not supported in, 870
  installer program, building, 1539–1541
  Interop Type Library, generating in, 1314–1316
  JavaScript debugger, 1132–1133
  .NET business objects, creating, 1296–1301
  RCW for COM component, creating, 1303–1304
  refactoring support in, 1591–1592
  resource files, 1395–1403
  server controls, creating ASP.NET, 1204
  triggers, building, 919–920
  UpdateProgress control, 920–925
  WCF service, building, 1362
  Web Part controls, creating custom, 844–845
  Web Site Administration Tool access, 809, 1500
  WebPartManager control, 814
  WebPartZone controls, dragging and dropping in, 817
  XAML, viewing and editing in, 1612–1617
  XHTML in, 1571–1573
  XML Web service. *See* XML Web Services, building
  zone layouts, 817
Visual Studio Conversion Wizard, 1574–1579
Visual Studio Toolbox, 932–934
visualizers, data, 1129
`.vsi` **file extension,** 931–932

W

W3C (World Wide Web Consortium), 500, 502
`w3wp.exe`**,** 1047, 1058
Wait approach
  multiple wait handles, 442–451
  overview of, 439–442
`WaitAny` **method, asynchronous processes,** 446–451
`WaitHandle` **class,** 435
`WaitOne` **method,** `WaitHandle` **class,** 439–442
`Warn()` **method,** `TraceContext` **class,** 1108
`WatermarkText` **property, TextBoxWatermarkExtender control,** 981
WCF (Windows Communication Foundation), 1360–1370
  building service, 1362–1363
  creating interface, 1365–1366
  creating service framework, 1364
  development of, 1360
  hosting service in console application, 1367–1368
  larger move to SOA, 1360–1361
  overview of, 1361–1362
  reviewing WSDL document, 1368–1370
  service components, 1363
  utilizing interface, 1366–1367
WCF (Windows Communication Foundation), building consumer application, 1370–1379
  adding service reference, 1370–1371
  building consumer, 1377–1378
  configuration file changes, 1372–1373
  namespaces, 1379
  working with data contracts, 1374–1377
  writing consumption code, 1373
  WSDL and schema for `HelloCustomer` service, 1378–1379
Web Developer Toolbar, Firefox, 1586–1587
Web event providers, 612–620
  `EventLogWebEventProvider`, 613–615
  `IiSWebEventProvider`, 619
  overview of, 612–613
  `SimpleMailWebEventProvider`, 615–617
  `SqlWebEventProvider`, 618–619
  `TemplatedMailWebEventProvider`, 617–618
  `TraceWebEventProvider`, 619
  `WmiWebEventProvider`, 619–620

Web events
  defined, 612
  health monitoring. *See* health monitoring
  tracing using, 1121–1122
Web Farm, session state configuration, 1418–1421
Web gardening, 1039
Web pages
  adding templated controls to, 1252
  adding user controls to, 1194–1195
  dynamically adding user controls to, 1198–1203
  limitations of using HTML for styling, 862
  sending e-mail from, 1189–1190
  using external style sheets in, 864–865
  using inline styles in, 866
  using internal style sheets in, 865–866
Web Parts
  allowing user to change mode of page, 820–833
  creating custom, 844–850
  modifying zones, 833–840
  overview of, 811–813
  using WebPartManager control, 813–814
  using WebPartZone control, 817–820
  verbs, 836–838
  working with zone layouts, 814–817
Web Parts, connecting, 850–860
  on ASP.NET page, 856–858
  building consumer Web Part, 854–856
  building provider Web Part, 851–854
  dealing with master pages when, 858–860
  overview of, 850–851
Web server
  customizing with IIS 7, 564–569
  using built-in, 2–3
Web server controls, 107–192
  AdRotator, 151–152
  BulletedList, 157–162
  Button, 115–118
  Calendar, 142–151
  CheckBox, 129–132
  CheckBoxList, 132–134
  DropDownList, 121–124
  FileUpload, 164–174
  HiddenField, 162–164
  HyperLink, 120–121
  Image, 138–139
  ImageButton, 119–120
  ImageMap, 189–191
  Label, 108–110
  LinkButton, 119
  ListBox, 125–129
  Literal, 110–111
  MultiView, 174–178
  overview of, 107–108
  Panel, 153–156
  Placeholder, 156
  RadioButton, 134–136
  RadioButtonList, 136–137
  Table, 139–141
  TextBox, 111–115
  using HTML server controls vs., 64–65
  View, 174–178
  visually removing items from collections, 124–125
  Wizard, 178–189
  Xml, 153
Web services. *See* XML Web services
Web Services Description Language. *See* WSDL (Web Services Description Language) files
Web Services Enhancements (WSE), 1346, 1360
Web Setup Project, Windows Installer, 1541
Web Site Administration Tool
  Application tab, 1510–1512
  configuration file editing, 1452–1453
  Home tab, 1501
  overview of, 1499–1500
  Provider tab, 1512–1514
  role management service with, 799–800
  Security Setup Wizard, 1502–1507
  Security tab, creating users, 1507–1508
  Security tab, managing access rules, 1510
  Security tab, managing roles, 1508–1509
  Security tab, managing users, 1508
  Security tab, overview of, 1501–1502
  using, 808
Web sites
  adding in IIS Manager, 575–576
  building with Portal Framework. *See* Portal Framework
  changing name of in IIS Manager, 1516
  membership management set up, 758–761
  online resources, 1628
  role management set up, 796–799
  running with multiple versions of .NET, 1439
Web user control, 1194–1195
`web.config` **file, configuring**
  `<authentication>` node setting, 997–998
  `<rules>`, 1483–1485
  `ActiveDirectoryMembershipProvider` in, 600–602
  administrator section, 716–717
  ASP.NET cache, 1081
  attribute values, 630–631
  `AuthorizationStoreRoleProvider`, 605–606
  configuration files. *See* configuration
  connection strings, 385–386, 595–598
  defined, 996
  `DpapiProtectedConfigurationProvider`, 621–622
  file-size limitation, 169–170
  Forms authentication. *See* Forms-based authentication
  health monitoring, 1479
  HttpHandler, 1294
  `HttpModule`, 1181, 1281–1282
  `InProcSessionStateStore`, 611
  localization, 711
  master page, 241–242
  multiple languages, 37–38
  personalization properties, 725–726
  provider instances, 641–642
  providers, 625

  role management, 716
  role management service, 798–799
  security trimming, 718–720
  SMTP setup for emailing Web events, 1494
  SQL Server cache invalidation, 1092–1093
  `SqlPersonalizationProvider`, 623–625
  `SqlSessionStateStore`, 612
  storing connection information, 315–317
  System.LINQ namespace, 465
  themes, 265, 267
  WCF consumer, 1372–1373
  Web event providers. *See* Web event providers
  Web site membership, 759–761
  Windows authentication. *See* Windows-based authentication
  `WindowsTokenRoleProvider`, 604–605
  `XmlSiteMapProvider`, 608–609
`web.config.comments` **file,** 1386–1387
`WebControl` **class,** 70, 1203–1209
`WebEventProvider` **class,** 613
weblogs (blogs), and `XmlDataSource` **control,** 535–537
weblogs (blogs), online resources, 1627
`WebMethod` **attribute**
  overloading, 1346–1349
  specifying SOAP header for, 1352–1353
  XML Web service, 1332–1336
WebPageTraceListener, 1116
`WebPart` **class**
  building consumer Web Part, 856
  building provider Web Part, 854
  creating custom Web Part control, 844–850
  working with, 843–844
`WebPartManager` **class,** 841
WebPartManager control
  adding Web Parts to page, 825–826
  connecting Web Parts, 833
  connecting Web Parts using, 856–858
  dealing with master pages, 859
  modifying Web Part settings, 828–833
  moving Web Parts, 826–828
  overview of, 813–814, 820–825
WebParts provider, 623–625
`<WebPartsTemplate>`**, DeclarativeCatalogPart control,** 835–836
`WebPartZone` **class,** 842–843
WebPartZone control
  connecting Web Parts on page, 857–858
  controlling Web Part verbs, 837–840
  working with, 817–820
`WebRequest` **class**
  defined, 1188–1189
  `FileWebRequest` class, 1188–1189
  `FtpWebRequest` class, 1186–1188
  `HttpWebRequest` class, 1183–1186
`WebResource.axd` **handler,** 1227
`WebResponse` **class**
  defined, 1188–1189
  `FileWebResponse` class, 1188–1189
  `FtpWebResponse` class, 1186–1188
  `HttpWebResponse` class, 1183–1186
`WebService` **attribute,** 1332
`@WebService` **directive,** 1329
`<WebServiceBinding>` **attribute,** 1330
`WebService.cs` **file,** 1329–1330
`WebService.vb`**,** 1329–1330
`Web.sitemap` **file**
  for localization, 710–711
  nesting, 720–722
  setting up administrators' section, 717
`Web.sitemap.fi.resx` **file,** 712–714
`Web.sitemap.resx` **file,** 712–714
`where` **clause**
  LINQ query, 469–471
  LinqDataSource control, 304–305
wildcards, IIS, 1286–1288
WinDiff, 1601
Windows Communication Foundation. ***See*** **WCF (Windows Communication Foundation)**
Windows Installer service
  application packaging and deploying, 1547–1550
  Custom Actions Editor, 1562–1564
  defined, 1539–1541
  File System Editor, 1554–1557
  File Types Editor, 1559–1561
  Launch Conditions Editor, 1568–1569
  Registry Editor, 1557–1559
  User Interface Editor, 1561–1562
  Windows Installer, 1550–1554
Windows NT Challenge/Response authentication, 1004, 1427–1428
Windows Presentation Foundation (WPF), 896
Windows Server, 2008, 565–568
Windows Services, 1418–1419
Windows Vista
  command-line setup options, 567–568
  configuration file editing tools, 1452–1453
  IIS Manager. *See* IIS (Internet Information Service) Manager
  installing IIS 7 on, 565
  using Basic authentication option in, 1005
  using IIS 7.0 Manager in, 1032
  working with file extensions, 1028
Windows XP, 1128
Windows-based authentication, 998–1006
  `<allow>` and `<deny>` nodes, 1001–1002
  authenticating/authorizing groups, 1002–1003
  authenticating/authorizing HTTP transmission method, 1003
  authenticating/authorizing users, 999–1001
  Basic authentication, 1004–1005
  configuring, 1427–1428
  creating users, 998–999

- **Windows-based authentication *(continued)***
  - defined, 997
  - Digest authentication, 1005–1006
  - encrypting, 1451–1452
  - Integrated Windows authentication, 1004
  - overview of, 998
  - Security Setup Wizard, 1502–1503
  - `WindowsIdentity` object, 1020–1023
- `WindowsBuiltInRole` **enumeration,** 1020–1021
- `WindowsIdentity` **object,** 1020–1023
- `WindowsTokenRoleProvider`**,** 604–606
- **WinMerge,** 1601
- **Wizard server control,** 178–189
  - adding header to, 181–182
  - `AllowReturn` attribute, 180
  - customizing side navigation, 180
  - events, 183–184
  - navigation system for, 182–183
  - overview of, 178–180
  - showing form elements with, 184–189
  - `StepType` attribute, 180–181
- **<WizardSteps> element,** 768
- `WmiWebEventProvider`**,** 613, 619–620
- **worker process, configuration settings,** 1438–1440
- **Worker Process Restarts performance counter,** 1470
- **Worker Processes Running performance counter,** 1470
- **workplace pane, IIS Manager,** 569
- **World Wide Web Consortium (W3C),** 500, 502
- **WPF (Windows Presentation Foundation),** 896
- `Wrap` **attribute, multiline text boxes,** 112
- `Write()` **method**
  - `Stream` class, 1169–1170
  - `TraceContext` class, 1108
  - writing to `MemoryStream`, 1170
- `WriteEntry()`**, writing to event log,** 1466
- `Writer` **class,** 1166, 1171–1176
- **writing events**
  - with event log, 1464–1468
  - routing events to SQL server, 1487–1490
  - via configuration, 1486–1487
- `Wrox.master` **file,** 238–239, 257–258
- `WroxOpera.master` **file,** 257–258
- ***Wrox's Beginning Web Programming with HTML, XHTML, and CSS* (Wiley Publishing, Inc.),** 72
- **WSDL (Web Services Description Language) files**
  - consuming XML Web services, 1337–1338
  - schema for `HelloCustomer` service, 1378–1379
  - WCF service linking to, 1368–1370
  - XML Web services linking to, 1334
- `.wsdl` **file extension,** 33–38
- **WSE (Web Services Enhancements),** 1346, 1360
- **WS-I Basic Profile spec,** 1347–1348
- **WS-Policy,** 1361
- **www.asp.net,** 1590–1591
- **WYSIWYG design, of Visual Studio,** 884

X

- **XAML**
  - converting vector content to, 1611–1613
  - editing in Notepad, 1620
  - Export plug-in, 1611
  - using XAML editor, 1617
  - viewing or editing with Expression Blend, 1618–1619
  - viewing or editing with Internet Explorer, 1614–1617
- **XCOPY, deploying with,** 1531–1534
- `XDocument`
  - processing XML with, 508–509
  - querying with `XPath`, 529
  - validating against schema with, 511–513
- **XHTML,** 1105, 1571–1572
- **XLINQ. *See* LINQ to XML (XLINQ)**
- **XML (eXtensible Markup Language), 497–556. *See also* AJAX (Asynchronous JavaScript and XML)**
  - acronym design and, 498
  - basics of, 498–500
  - binding Menu server control to, 701–702
  - binding TreeView control to, 676–679
  - configuration files, 1410
  - creating CLR objects from, 517–518
  - creating with `XmlWriter`, 518–521
  - data binding, 369
  - and databases, `FOR XML AUTO` clause, 545–549
  - DataSets, 530–533
  - integration with, 1326
  - LINQ to. *See* LINQ to XML (XLINQ)
  - `NameTable` optimization, 513–515
  - overview of, 497–498
  - retrieving .NET CLR types from, 515–516
  - schema, editing, 503–506
  - schema, using with `XmlTextReader`, 509–511
  - schema, validating against with `XDocument`, 511–513
  - serialization, 516–517
  - site maps based on, 662–663
  - using AdRotator control, 151–152
  - using `XDocument` vs. `XmlReader`, 508–509
  - `XML` data type and SQL Server 2005, 549–556
  - Xml InfoSet, 500–501
  - `XmlDataSource` control, 533–537
  - `XmlDocument` and `XPathDocument`, 525–529
  - `XmlReader` and `XmlWriter`, 506–508, 524–525
  - XSD, 501–502
  - XSLT, 537–544

XML Editor, 503–506, 1104–1106
Xml InfoSet, 500–501
XML Literals, 522
XML Schema Definition. ***See*** **XSD (XML Schema Definition)**
XML Schema Editor, 505
XML server control, 153
XML Web services, 1325–1379
  caching responses, 1349–1350
  communication between disparate systems, 1325–1327
  overloading Web methods, 1346–1349
  overview of, 1325
  SOAP headers, 1350–1357
  Windows Communication Foundation for. *See* WCF (Windows Communication Foundation)
XML Web services, building, 1327–1336
  contents of `Service.asmx` file, 1328–1329
  exposing custom datasets as SOAP, 1330–1333
  interface, 1333–1336
  overview of, 1327–1328
  `WebService.vb` or `WebService.cs` file, 1329–1330
XML Web services, consuming
  adding Web reference, 1336–1338
  asynchronously, 1357–1360
  invoking from client application, 1338–1341
  overview of, 1336
  transport protocols for, 1341–1346
  using SOAP headers, 1353–1355
`XmlConvert` **class,** 515
`XmlDataDocument` **class,** 531–533
`XmlDataSource` **server control**
  binding TreeView control to XML file using, 676–679
  data binding and, 162
  overview of, 307–309, 533–537
`XmlDocument`
  databases and XML, 545–549
  `XDocument` vs., 508–509
  `XPathDocument` and, 525–529
XmlHTTP, 95–96
`XMLHttpRequest` **object,** 895, 898–899
`XmlMembershipProvider` **class**
  constructing class skeleton, 636–640
  creating CustomProviders application, 635–636
  creating XML user data store, 640
  defining provider instance in web.config file, 641–642
  implementing methods/properties of, 643–646
  not implementing methods/properties of, 642–643
`XmlNamespaceManager`**,** 533–534
`XmlReader`
  calling `Close()` when done with, 548–549
  improvements in 2.0, 524–525
  optimizing with `NameTable`, 513–515
  overview of, 506–508
  `ReadSubTree` and XML serialization, 516–517
  retrieving .NET CLR types from, 515–516
  `XDocument` vs., 508–509
`XmlReaderSettings` **object,** 506–508
`XmlSerializer`**,** 516–517, 523–524
XmlSiteMapProvider, 608–609, 718–720
`XmlTextReader`**,** 509–511
`XmlWriter`**,** 506–508, 518–521, 524–525
`XmlWriterSettings` **class,** 519–521
XmlWriterTraceListener, 1118–1119
`XPath`**,** 529, 533–534
`XPathBinder` **class, XML data binding,** 369
`XPathDocument`**,** 525–529, 538
`XPathNavigator` **class,** 525–528
`XPathNodeIterator`**,** 526–529
XSD (XML Schema Definition)
  LINQ to XSD, 518
  overview of, 501–502
  working with WCFservice, data contracts, 1374
  WSDL and, 1378–1379
  XML Editor as default view for, 505
XSD Designer, 505
`XslCompiledTransform` **class,** 538–541, 543–544
XSLT, 537–544
  databases and, 548–549
  debugger, 543–544
  overview of, 537–538
  speed of, 538
  `XslCompiledTransform` class, 538–541
  `XSLTC.exe` command-line compiler, 539–543
`XSLTC.exe` **command-line compiler,** 539–543
`XSLTCommand`**,** 542
`XsltSettings` **object,** 538–539

Y

YSlow, by Yahoo!, 1585–1586

Z

zones, Web
  adding Web Parts to page, 825–826
  defining, 813
  layouts, 814–817
  modifying, 833–840
  moving Web Parts between, 826–828
  using WebPartZone control, 817–820
  WebPartManager control managing, 813–814
`ZoneTemplate` **attribute, WebPartZone control,** 819–820